

# The Packing While Traveling Problem

S. Polyakovskiy<sup>a,\*</sup>, F. Neumann<sup>a</sup>

<sup>a</sup>*School of Computer Science  
The University of Adelaide  
Adelaide, SA 5005, Australia.*

---

## Abstract

This paper introduces the Packing While Traveling problem as a new non-linear knapsack problem. Given are a set of cities that have a set of items of distinct profits and weights and a vehicle that may collect the items when visiting all the cities in fixed order. Each selected item contributes its profit, but produces a transportation cost relative to its weight. The problem asks to find a subset of the items such that the total gain is maximized. We investigate constrained and unconstrained versions of the problem and show that both are  $\mathcal{NP}$ -hard. We propose a pre-processing scheme that decreases the size of instances making them easier for computation. We provide lower and upper bounds based on mixed-integer programming (MIP) adopting the ideas of piecewise linear approximation. Furthermore, we introduce two exact approaches: one is based on MIP employing linearization technique, and another is a branch-infer-and-bound (BIB) hybrid approach that compounds the upper bound procedure with a constraint programming model strengthened with customized constraints. Our experimental results show the effectiveness of our exact and approximate solutions in terms of solution quality and computational time.

*Keywords:* Non-linear knapsack problem, mixed-integer programming, constraint programming, linearization technique, piecewise approximation, hybrid optimization, branch-infer-and-relax, NP-hardness.

---

## 1. Introduction

Generally, the traditional statements of routing problems studied in the operations research literature base the computation of transportation costs on a linear function. However, in real practice, it might be necessary to deal with costs that have a nonlinear nature. For example, the study on the factors affecting truck fuel economy published by GOODYEAR (2008) reveals that vehicle miles per gallon decreases as gross combination weight increases assuming speed

---

\*Corresponding author

*Email addresses:* sergey.polyakovskiy@adelaide.edu.au (S. Polyakovskiy), frank.neumann@adelaide.edu.au (F. Neumann)

is maintained constant. In other words, a heavily loaded truck will use much more fuel than a lightly loaded one, and this relation is not linear. In recent years, the research on dependence of fuel consumption on different factors, like a travel velocity, a load's weight, and vehicle's technical specifications, in various Vehicle Routing Problems (VRP) has gained attention from the operations research community. Mainly, this interest is motivated by a wish to be more accurate with the evaluation of transportation costs, and therefore to stay closer to reality. Indeed, an advanced precision would immediately benefit to transportation efficiency measured by the classic petroleum-based costs and the novel greenhouse gas emission costs. Furthermore, the proper estimation of costs and its computational simplicity should evolve optimization approaches and enhance their performance. In VRP in general, and in the Green Vehicle Routing Problems (GVRP) that consider energy consumption in particular, given are a depot and a set of customers that are to be served by a set of vehicles collecting (or delivering) required items. While the set of items is fixed, the goal is to find a route for each vehicle such that the total size of assigned items does not exceed the vehicle's capacity and the total transportation cost over all vehicles is minimized. We refer to the survey of Lin et al. (2014) for an extended overview on VRP and GVRP.

Oppositely to VRP and GVRP, we consider the situation of a single vehicle whose route is given, but items can be either collected or skipped. This situation gives rise to a problem that we designate as Packing While Traveling (PWT). In PWT, the items are distributed among the cities. The vehicle visits all the cities in a specific order and collects the items of its choice. Each item has a profit and a weight, and the vehicle may collect any unless the total weight of chosen items exceeds the vehicle's capacity. The vehicle travels between two cities with velocity that depends on the weight of the items collected in all the previously attended cities. Being selected, an item contributes its profit to the overall reward. However, its weight slows down the vehicle. This leads to a transportation cost depended on a traveling time, and therefore has a negative impact on the reward. The problem asks to find a packing plan that minimizes the difference between the total profit of the selected items and their transportation cost. PWT arises in some practical applications. For example, a supplier having a single truck has to decide on goods to purchase going through a fixed route in order to maximize profitability of future sales.

PWT originates from the Traveling Thief Problem (TTP) introduced by Bonyadi et al. (2013). TTP combines the classical Traveling Salesperson Problem (TSP) with the 0-1 Knapsack Problem (KP) and allows permutation of the order of the cities. PWT uses the same cost function as the TTP, and the only difference is the assumption of a fixed route. In this sense, any approach to PWT can also be applied to TTP as a subroutine to solve its packing component. Vansteenwegen et al. (2011) give a review on the so-called orienteering problem that is somehow related to TTP. There a set of vertices is given, each with a score, and the goal is to determine a path, limited in length, that visits some vertices and maximizes the sum of the collected scores. Feillet et al. (2005) present a classification of traveling salesman problems with profits (TSPs with

profits) and survey the existing literature on this field. TSPs with profits are a generalization of TSP, where it is not necessary to visit all vertices. A profit is associated with each vertex. The overall goal is the simultaneous optimization of the collected profit and the travel costs. In this sense, TTP has some relation to the Prize Collecting TSP Balas (1989) where a decision is made on whether to visit a given city. In the Prize Collecting TSP, a city-dependent reward is obtained when a city is visited and a city-dependent penalty has to be paid for each non-visited city. In contrast to this, the TTP requires that each given city is visited. Furthermore, each city has a set of available items with weights and profits and a decision has to be made on which items to pick. TTP also relates to the traveling salesman subtour problem studied by Westerlund et al. (2006) where given is an undirected graph with edge costs and both revenues and weights on the vertices, and the goal is to find a subtour that includes a depot vertex, satisfies a knapsack constraint on the vertex weights, and that minimizes edge costs minus vertex revenues along the subtour.

In substance, PWT considers a trade-off between the profits of collected items and the transportation cost affected by their total weight. It represents a class of nonlinear knapsack problems. Knapsack problems belong to the core combinatorial optimization problems and have been frequently studied in the literature from the theoretical as well as experimental perspective (Garey & Johnson (1979); Martello & Toth (1990)). While the classical knapsack problem asks for maximization of a linear pseudo-Boolean function under one linear constraint, different generalizations and variations have been investigated such as the multiple knapsack problem (Chekuri & Khanna (2005)) and multi-objective knapsack problems (Erlebach et al. (2001)). Furthermore, knapsack problems with nonlinear objective functions have been studied in the literature from different perspectives (Bretthauer & Shetty (2002)). Hochbaum (1995) considers the problem of maximizing a separable concave objective function subject to a packing constraint and provided an FPTAS. An exact approach for a nonlinear knapsack problem with a nonlinear term penalizing the excessive use of the knapsack capacity has been given by Elhedhli (2005).

The preliminary version of our study on PWT has appeared in Polyakovskiy & Neumann (2015). Here, we significantly improve our earlier results. We introduce an upper bound technique that along with the enhanced pre-processing scheme allows to solve a larger range of the instances to optimality and to dramatically decrease running times. Furthermore, we introduce a hybrid approach that combines constraint programming with the upper bound procedure. It is superior on many test instances and produces optimal results in a very short time. The rest of the paper is organized as follows. We give the formal statement of PWT in Section 2 and discuss its complexity in Section 3. Section 4 addresses sequencing constraints that are repeatedly used later on in our approaches. In Section 3, we provide a pre-processing scheme which allows to identify unprofitable and compulsory items, and therefore decrease the size of the PWT's instances. Section 6 explains lower and upper bound techniques. In Sections 7 and 8, we introduce our two exact approaches: one that is based on MIP, and a hybrid one that adopts a branch-infer-and-bound paradigm. Fi-

nally, we report on the results of our experimental investigations in Section 9 and finish with some conclusions.

## 2. Problem Statement

The Packing While Traveling (PWT) problem can be formally stated as follows. Given is a route  $N = (1, 2, \dots, n+1)$  as a sequence of  $n+1$  unique cities and a set  $M$  of  $m$  items distributed among first  $n$  cities. Distance  $d_i > 0$  between two consecutive cities  $(i, i+1)$  is known, for any  $1 \leq i \leq n$ . Every city  $i$  contains a set of distinct items  $M_i = \{e_{i1}, \dots, e_{im_i}\}$ ,  $M = \cup_{i=1}^n M_i$ . Each item  $e_{ik} \in M$  has a positive integer profit  $p_{ik}$  and a weight  $w_{ik}$ . There is a vehicle that visits all the cities in the order of a route  $N$ . The vehicle may collect any item in any city unless the total weight of selected items exceeds its capacity  $W$ . Collecting an item  $e_{ik}$  leads to a profit contribution  $p_{ik}$ , but increases the transportation cost as the weight  $w_{ik}$  slows down the vehicle. The vehicle travels along  $(i, i+1)$  with velocity  $v_i \in [v_{\min}, v_{\max}]$  which depends on the weight of the items collected in the first  $i$  cities. When the vehicle is empty, it runs with its maximal velocity  $v_{\max}$ . And vice versa, it runs with minimal velocity  $v_{\min} > 0$  when is completely full. The objective is to find a subset of  $M$  such that the difference between the profit of the selected items and the transportation cost is maximized.

To state the problem precisely, we give a nonlinear binary integer program formulation. Let a binary decision vector  $x \in \{0, 1\}^m$  represent a solution of the problem such that  $x_{ik} = 1$  iff  $e_{ik}$  is selected. Then the travel time  $t_i = \frac{d_i}{v_i}$  along  $(i, i+1)$  is the ratio of the distance  $d_i$  and the current velocity

$$v_i = v_{\max} - \nu \sum_{j=1}^i \sum_{k=1}^{m_j} w_{jk} x_{jk}$$

which is determined by the weight of the items collected in cities  $1, \dots, i$ . The value  $\nu = \frac{v_{\max} - v_{\min}}{W}$  is constant and defined by the input parameters. The velocity depends on the weight of the chosen items linearly. The overall transportation cost is given by the sum of the transportation costs along all the edges  $(i, i+1)$ ,  $1 \leq i \leq n$ , multiplied by a given rent rate  $R > 0$ . In summary, the problem is given by the following nonlinear binary program (PWT<sup>c</sup>):

$$\max \sum_{i=1}^n \left( \sum_{k=1}^{m_i} p_{ik} x_{ik} - \frac{R d_i}{v_{\max} - \nu \sum_{j=1}^i \sum_{k=1}^{m_j} w_{jk} x_{jk}} \right) \quad (1)$$

$$\begin{aligned} \text{s.t. } & \sum_{i=1}^n \sum_{k=1}^{m_i} w_{ik} x_{ik} \leq W \\ & x_{ik} \in \{0, 1\}, e_{ik} \in M \end{aligned} \quad (2)$$

Here,  $1$  is a non-monotone submodular function.

We also consider the unconstrained version  $\text{PWT}^u$  of  $\text{PWT}^c$  where  $W \geq \sum_{e_{ik} \in M} w_{ik}$  such that every selection of items yields a feasible solution. Given a real value  $B$ , the decision variant of  $\text{PWT}^c$  and  $\text{PWT}^u$  has to answer the question whether the value of (1) is at least  $B$ .

### 3. Complexity of the Problem

In this section, we investigate the complexity of  $\text{PWT}^c$  and  $\text{PWT}^u$ .  $\text{PWT}^c$  is NP-hard as it is a generalization of the classical NP-hard 0-1 knapsack problem (Martello & Toth (1990)). In fact, assigning zero either to the rate  $R$  or to every distance value  $d_i$  in  $\text{PWT}^c$ , we obtain KP. We demonstrate that in contrast to KP the unconstrained version  $\text{PWT}^u$  of the problem remains  $\mathcal{NP}$ -hard. We show this by reducing the  $\mathcal{NP}$ -complete *subset sum problem* (SSP) to the decision variant of  $\text{PWT}^u$  which asks whether there is a solution with objective value at least  $B$ . The input for SSP is given by  $m$  positive integers  $S = \{s_1, \dots, s_m\}$  and a positive integer  $Q$ . The question is whether there exists a vector  $x \in \{0, 1\}^m$  such that  $\sum_{k=1}^m s_k x_k = Q$ .

**Theorem 1.**  *$\text{PWT}^u$  is  $\mathcal{NP}$ -hard.*

PROOF. We start with encoding the instance of SSP given by the set of integers  $S$  and the integer  $Q$  as the instance  $I$  of  $\text{PWT}^u$  having two cities. The first city contains all the  $m$  items while the second city is a destination point free of items. We set the distance between two cities as  $d_1 = 1$ , the capacity of the vehicle as  $W = \sum_{k=1}^m s_k$ , and set  $p_{1k} = w_{1k} = s_k$ ,  $1 \leq k \leq m$ . Subsequently, we set  $v_{max} = 2$  and  $v_{min} = 1$  which implies  $\nu = 1/W$  and define  $R^* = W(2 - Q/W)^2$ .

Consider the nonlinear function  $f_{R^*}: [0, W] \rightarrow \mathbb{R}$  defined as

$$f_{R^*}(w) = w - \frac{R^*}{2 - w/W}. \quad (3)$$

$f_{R^*}$  defined on the interval  $[0, W]$  is a continuous convex function that reaches its unique maximum in the point  $w^* = W \cdot (2 - \sqrt{R^*/W}) = Q$ , i.e.  $f_{R^*}(w) < f_{R^*}(w^*)$  for  $w \in [0, W]$  and  $w \neq w^*$ . Then  $f_{R^*}(Q)$  is the maximum value for  $f_{R^*}$  when being restricted to integer input, too. Therefore, we set  $B = f_{R^*}(Q)$  and the objective function for  $\text{PWT}^u$  is given by

$$g_{R^*}(x) = \sum_{k=1}^m p_{1k} x_k - \frac{R^*}{2 - \frac{1}{W} \sum_{k=1}^m w_{1k} x_k}. \quad (4)$$

There exists an  $x \in \{0, 1\}^m$  such that  $g_{R^*}(x) \geq B = f_{R^*}(Q)$  iff  $\sum_{k=1}^m s_k x_k = \sum_{k=1}^m w_{1k} x_k = \sum_{k=1}^m p_{1k} x_k = Q$ . Therefore, the instance of SSP has answer YES iff the optimal solution of the  $\text{PWT}^u$  instance  $I$  has objective value at least  $B = f_{R^*}(Q)$ . Obviously, the reduction can be carried out in polynomial time which completes the proof.  $\square$

#### 4. Sequencing Constraints

In this section, we derive a set of constraints that speed up the reasoning to be done within our algorithms. Specifically, the constraints establish priority among the items positioned in the same or different cities of the tour. The first portion of the constraints results from the fact that item  $e_{il}$  in city  $i$  should not be selected prior to another item  $e_{ik}$  positioned in the same city when the condition  $(p_{il} < p_{ik}) \wedge (w_{il} \geq w_{ik})$  holds. This constraint (SC<sup>i</sup>) has the form of

$$x_{il} \leq x_{ik}, l \neq k, e_{il}, e_{ik} \in M_i : (p_{il} < p_{ik}) \wedge (w_{il} \geq w_{ik}).$$

Let  $\Delta_l^{ji}$  denote a lower bound on the cost of transportation of item  $e_{jl}$  from city  $j$  to succeeding city  $i$  computed as

$$\Delta_l^{ji} = R \sum_{a=j}^{i-1} d_a \left( \frac{1}{v_{max} - \nu \left( w_{jl} + \sum_{b=1}^a w_b^c \right)} - \frac{1}{v_{max} - \nu \sum_{b=1}^a w_b^c} \right),$$

where  $w_b^c$  is the total weight of the compulsory items collected in city  $b$ . Compulsory items must be a part of an optimal solution (see Section 5 for details). Then another set of constraints use the fact that item  $e_{jl}$  in city  $j$  should not be selected prior to item  $e_{ik}$  in city  $i$  such that the condition  $(p_{jl} - \Delta_l^{ji} < p_{ik}) \wedge (w_{jl} \geq w_{ik})$  holds. This constraint (SC<sup>ii</sup>) takes the form of

$$x_{jl} \leq x_{ik}, j < i, e_{jl} \in M_j, e_{ik} \in M_i : (p_{jl} - \Delta_l^{ji} < p_{ik}) \wedge (w_{jl} \geq w_{ik}).$$

Similarly, let  $\overline{\Delta}_l^{ji}$  denote an upper bound on the cost of transportation of item  $e_{jl}$  from city  $j$  to succeeding city  $i$  computed as

$$\overline{\Delta}_l^{ji} = R \sum_{a=j}^{i-1} d_a \left( \frac{1}{v_{max} - \nu \cdot \min \left( \sum_{b=1}^a w_b^{max}, W \right)} - \frac{1}{v_{max} - \nu \left( \min \left( \sum_{b=1}^a w_b^{max}, W \right) - w_{jl} \right)} \right),$$

where  $w_b^{max}$  is the total weight of all the items existing in city  $b$ . Then one more set of constraints arises from the fact that item  $e_{ik}$  in city  $i$  should not be selected prior to item  $e_{jl}$  in city  $j$  such that  $(p_{jl} - \overline{\Delta}_l^{ji} > p_{ik}) \wedge (w_{jl} \leq w_{ik})$  holds. This constraint (SC<sup>iii</sup>) has the following form:

$$x_{jl} \geq x_{ik}, j < i, e_{jl} \in M_j, e_{ik} \in M_i : (p_{jl} - \overline{\Delta}_l^{ji} > p_{ik}) \wedge (w_{jl} \leq w_{ik}).$$

#### 5. Pre-processing

In this section, we introduce a pre-processing scheme to identify items of a given instance  $I$  that can be either directly included or discarded. Excluding such items from solution process can significantly speed up algorithms. We distinguish between two kinds of items that are identified in the pre-processing: *compulsory* and *unprofitable* items. We call an item *compulsory* if its inclusion

in any feasible solution increases the objective function value, and call an item *unprofitable* if it does not do that. Therefore, an optimal solution must contain all compulsory items while all unprofitable items must be discarded. In order to identify *compulsory* and *unprofitable* items, we consider the total transportation cost that a set of items produces.

**Definition 1 (Total Transportation Cost).** Let  $O \subseteq M$  be a subset of items. We define the total transportation cost along route  $N$  when the items of  $O$  are selected as

$$t_O = R \cdot \sum_{i=1}^n \frac{d_i}{v_{max} - \nu \sum_{j=1}^i \sum_{e_{jk} \in O_j} w_{jk}},$$

where  $O_j = M_j \cap O$ ,  $1 \leq j \leq n$ , is the subset of  $O$  selected in city  $j$ .

Based on the given instance  $I$ , we can identify unprofitable items for  $PWT^c$  according to the following proposition.

**Proposition 1 (Unprofitable Item,  $PWT^c$  Case).** *Let  $I$  be an arbitrary instance of  $PWT^c$ . If  $p_{ik} \leq R(t_{\{e_{ik}\}} - t_\emptyset)$ , then  $e_{ik}$  is an unprofitable item.*

PROOF. We assume that  $p_{ik} \leq R(t_{\{e_{ik}\}} - t_\emptyset)$  holds. Let  $M^* \subseteq M \setminus \{e_{ik}\}$  denote an arbitrary subset of items excluding  $e_{ik}$  such that  $w_{ik} + \sum_{e_{jl} \in M^*} w_{jl} \leq W$  holds. We consider  $t_{M^* \cup \{e_{ik}\}}$  and  $t_{M^*}$ . Since the velocity depends linearly on the weight of collected items and the travel time  $t_i = d_i/v_i$  along  $(i, i+1)$  depends inversely proportional on the velocity  $v_i$ , the inequality  $(t_{\{e_{ik}\}} - t_\emptyset) \leq (t_{M^* \cup \{e_{ik}\}} - t_{M^*})$  holds. Therefore,  $p_{ik} \leq R(t_{M^* \cup \{e_{ik}\}} - t_{M^*})$  holds for any  $M^* \subseteq M \setminus \{e_{ik}\}$  that completes the proof.  $\square$

Proposition 1 helps to determine whether the profit  $p_{ik}$  of item  $e_{ik}$  is large enough to cover the least incremental transportation cost it incurs when selected in the packing plan  $x$ . In this case, the least incremental transportation cost results from accepting the selection of  $e_{ik}$  as only selected item in  $x$  versus accepting empty  $x$  as a solution. It is important to note that Proposition 1 can reduce  $PWT^c$  problem to  $PWT^u$  by excluding items so that the sum of the weights of all remaining items does not exceed the weight bound  $W$ . In this case, we can further refine the set of items by searching for those ones that must be a part of any solution of  $PWT^c$ . We identify compulsory items for the unconstrained case according to the following proposition.

**Proposition 2 (Compulsory Item).** *Let  $I$  be an arbitrary instance of  $PWT^u$ . If  $p_{ik} > R(t_M - t_{M \setminus \{e_{ik}\}})$ , then  $e_{ik}$  is a compulsory item.*

PROOF. We work under the assumption that  $p_{ik} > R(t_M - t_{M \setminus \{e_{ik}\}})$  holds. In the case of  $PWT^u$ , all the existing items can fit into the vehicle at once and all subsets  $O \subseteq M$  are feasible. Let  $M^* \subseteq M \setminus \{e_{ik}\}$  be an arbitrary subset of items excluding  $e_{ik}$ , and consider  $t_{M \setminus M^*}$  and  $t_{M \setminus M^* \cup \{e_{ik}\}}$ , respectively. Since the velocity depends linearly on the weight of collected items and the travel time  $t_i = d_i/v_i$  along  $(i, i+1)$  depends inversely proportional on the

velocity  $v_i$ , we have  $(t_M - t_{M \setminus \{e_{ik}\}}) \geq (t_{M \setminus M^*} - t_{M \setminus M^* \setminus \{e_{ik}\}})$ . This implies that  $p_{ik} > R(t_{M \setminus M^*} - t_{M \setminus M^* \setminus \{e_{ik}\}})$  holds for any subset  $M \setminus M^*$  of items which completes the proof.  $\square$

For the unconstrained variant  $\text{PWT}^u$ , Proposition 2 is valid to determine whether item  $e_{ik}$  is able to cover by its  $p_{ik}$  the largest possible incremental transportation cost it may generate when has been selected in  $x$ . Here, the largest possible incremental transportation cost is computed in respect to the worst case scenario when all the possible items are selected along with  $e_{ik}$ , and therefore the vehicle has the maximal possible load and the least velocity, versus accepting all the items but  $e_{ik}$ . Having all compulsory items included in the unconstrained case according to Proposition 2, we can identify further unprofitable items. This is the case, as the inclusion of compulsory items already increases the travel time and therefore reducing the positive contribution to the overall objective value. We find unprofitable items for  $\text{PWT}^u$  in respect to the following proposition.

**Proposition 3 (Unprofitable Item,  $\text{PWT}^u$  Case).** *Let  $I$  be an arbitrary instance of  $\text{PWT}^u$  and  $M^c$  be the set of all compulsory items. If  $p_{ik} \leq R(t_{M^c \cup \{e_{ik}\}} - t_{M^c})$ , then  $e_{ik}$  is an unprofitable item.*

PROOF. We assume that  $p_{ik} \leq R(t_{M^c \cup \{e_{ik}\}} - t_{M^c})$  holds. Recall that in the case of  $\text{PWT}^u$ , all the existing items can fit into the vehicle at once and all subsets  $O \subseteq M$  are feasible. Let  $M^* \subseteq M \setminus \{M^c \cup \{e_{ik}\}\}$  be an arbitrary subset of  $M$  that does not include any item of  $M^c \cup \{e_{ik}\}$  and consider  $t_{M^c \cup M^*}$  and  $t_{M^c \cup M^* \cup \{e_{ik}\}}$ . Since the velocity depends linearly on the weight of collected items and the travel time  $t_i = d_i/v_i$  along  $(i, i+1)$  depends inversely proportional on the velocity  $v_i$ , we have  $(t_{M^c \cup \{e_{ik}\}} - t_{M^c}) \leq (t_{M^c \cup M^* \cup \{e_{ik}\}} - t_{M^c \cup M^*})$ . Hence, we have  $p_{ik} \leq R(t_{M^c \cup M^* \cup \{e_{ik}\}} - t_{M^c \cup M^*})$  for any  $M^* \subseteq M \setminus \{M^c \cup \{e_{ik}\}\}$  which completes the proof.  $\square$

Proposition 3 determines for  $\text{PWT}^u$  whether the profit  $p_{ik}$  of item  $e_{ik}$  is large enough to cover the least incremental transportation cost resulted from its selection along with all known compulsory items. Specifically, in Proposition 3 the list incremental transportation cost follows from accepting the selection of  $e_{ik}$  along with the set of compulsory items  $M^c$  in  $x$  versus accepting just the selection of  $M^c$  as a solution.

It takes only a linear time to check any instance of  $\text{PWT}^c$  for unprofitable items in respect to Proposition 1. In fact, each item  $e_{ik}$  can be checked in a constant time if the total length of the path from city  $i$  to city  $n+1$  is known. When dealing with  $\text{PWT}^u$ , Propositions 2 and 3 can be applied iteratively to the remaining set of items until no compulsory or unprofitable item is found. The running time of all the rounds of the search is bounded by  $\mathcal{O}(nm^2)$ . Our preliminary investigation has shown that it is rather time-consuming to solve large and even moderate-sized unconstrained instances due to the time spent on computing the incremental transportation cost for each of the items separately as the Propositions 2 and 3 advise. Indeed, we cannot perform the



pre-processing step reasonably fast in respect to the time limits we apply in Section 9. Obviously, slow pre-processing can easily stultify all benefits of its use. To manage this, we use the reasoning similar to one that the sequencing constraints adopt in Section 4. Specifically, we deduce whether item  $e_{ik}$  is compulsory or unprofitable from the answer concerning item  $e_{jl}$  for which it has been already obtained. Algorithm 1 sketches the pseudocode of the enhanced algorithm, which runs in  $\mathcal{O}(m^3)$ , but operates up to two orders of magnitude faster in practice and allows to handle the largest instances of the test suite (see Section 9 for details).

The pre-processing algorithm works as follows. The loop (5-22) searches for compulsory items and the loop (26-47) searches for unprofitable ones. Once either no compulsory or no unprofitable item has been found within the corresponding loop, the algorithm terminates (cf. lines 23 and 48). We use two Boolean variables  $\mu_{ik}^u$  and  $\mu_{ik}^c$  that take value *true* to mark item  $e_{ik}$  as unprofitable and compulsory, respectively. Both values are initialized as  $\mu_{ik}^u = \mu_{ik}^c = \text{false}$ . Subsequently, variable  $\bar{w}_i^{max} = \sum_{b=1}^i w_b^{max}$  computes the maximal possible weight of the items that can be collected in city  $i$  and in all the preceding cities. Similarly, variable  $\bar{w}_i^c = \sum_{b=1}^i w_b^c$  computes the total weight of compulsory items existing in city  $i$  and in all the preceding cities. We use  $\bar{w}_i^{max}$  and  $\bar{w}_i^c$  to calculate, respectively, the largest possible incremental cost  $c_{max}$  and the minimal possible incremental cost  $c_{min}$  for each of the items in the loops of our algorithm (cf. lines 7-22 and lines 28-47). Furthermore, to make reasoning on the properties of item  $e_{ik}$  in respect to the known properties of item  $e_{jl}$ , we introduce the dummy profit  $p'_{jl}$  of  $e_{jl}$ . Specifically, when item  $e_{jl}$  has been shown to be either compulsory or not, or either unprofitable or not,  $p'_{jl}$  defines how large or small profit  $p_{ik}$  must be in respect to computed  $c_{max}$  or  $c_{min}$  to let  $e_{ik}$  have the same property as  $e_{jl}$  (cf. lines 11, 14, 32, and 35). For example, when item  $e_{ik}$  is proved to be compulsory independently of any another item (cf. line 22), its  $p'_{ik}$  is set to  $c_{max}$  (cf. line 21). This means that  $e_{ik}$  would be compulsory even if its profit was less than  $p_{ik}$ , but mainly greater than  $c_{max}$ . Therefore, to become a compulsory item as  $p_{ik}$  is, another item, say  $e_{i'k'}$  in city  $i' : i' \leq i$ , should have a weight that is smaller or equal to  $w_{ik}$  and its profit  $p_{i'k'}$  minus the corresponding largest possible incremental cost must be strictly larger than  $p'_{ik}$  (cf. line 14). Similarly, when  $e_{ik}$  is proved to be not a compulsory item independently of any other item, its  $p'_{ik}$  is also set to  $c_{max}$  (cf. line 21). This is because  $e_{ik}$  would not be compulsory even if its profit was larger than  $p_{ik}$ , but mainly less or equal to  $c_{max}$ . Therefore, to stay as not a compulsory item as  $p_{ik}$  is, another item, say  $e_{i'k'}$  in city  $i' : i' \leq i$ , should have a weight that is greater or equal to  $w_{ik}$  and its profit  $p_{i'k'}$  minus the corresponding least possible incremental cost must be at most  $p'_{ik}$  (cf. line 11). The same reasoning is to be done for the case when  $e_{ik}$  is proved as a compulsory item according to already known compulsory item  $e_{jl}$ . Here,  $p'_{ik}$  is set to  $p'_{jl} + c_{max}$  where  $c_{max}$  plays a role of the largest cost of transportation of  $e_{ik}$  from city  $i$  to succeeding city  $j$  (cf. line 16). In such a way, when considering other items in the future iterations, they are compared to item  $e_{jl}$  through the current item  $e_{ik}$  since  $e_{ik}$  implicitly

points to  $e_{jl}$  using the assigned dummy profit  $p'_{ik}$ . The same reasoning is valid for proving  $e_{ik}$  to be not a compulsory item according to already known not compulsory item  $e_{jl}$  where  $p'_{ik}$  is set to  $p'_{jl} + c_{min}$  (cf. line 12). In a similar way, we proceed with deduction of unprofitable items in the loop (26-47). Utilizing the dummy profits of items significantly strengthen deductions by relating the items to one of the items' group rapidly. Before applying our approaches given in Section 6, Section 7, and 8, we remove all unprofitable and compulsory items from the set  $M$  using these pre-processing steps.

## 6. Lower and Upper Bounds

In practice, approximation of nonlinear terms is an efficient way to deal with them. Although an approximate solution is likely to be different from an exact one, it might be close enough and obtainable in a reasonable computational time. In this section, we propose lower and upper bound techniques based on mixed-integer programming (MIP) adopting the ideas of piecewise linear approximation.

### 6.1. Lower Bound

Consider an arbitrary edge  $(i, i + 1)$  and the traveling time  $t'_i \in [t_{min}, t_{max}]$  per distance unit along it, for any  $i = 1, \dots, n$ . Here,  $t_{min} = 1/v_{max}$  and  $t_{max} = 1/v_{min}$  bound  $t'_i$  from below and from above, respectively. We partition the interval  $[t_{min}, t_{max}]$  into  $\lambda$  equal-sized sub-intervals and determine thus a set  $T = \{\tau_1, \dots, \tau_\lambda\}$  of straight line segments to approximate the curve of the function  $t(v)$  as illustrated in Figure 1a. Each segment  $\tau \in T$  is characterized by its minimal velocity  $v_\tau^{min}$  and its corresponding maximum traveling time per distance unit  $t_\tau^{max}$ , and by its maximum velocity  $v_\tau^{max}$  and its corresponding minimum traveling time per distance unit  $t_\tau^{min}$ . Specifically,  $(v_\tau^{min}, t_\tau^{max})$  and  $(v_\tau^{max}, t_\tau^{min})$  are the endpoints of segment  $\tau$  referred to as breakpoints. We approximate  $t'_i$  by the linear combination of  $t_\tau^{min}$  and  $t_\tau^{max}$  if  $v_i \in [v_\tau^{min}, v_\tau^{max}]$ .

Our lower bound MIP-based model uses three types of variables in addition to the binary decision variable  $x_{ik}$  for each item  $e_{ik} \in M$  from Section 2. Let  $w_i$  be a real variable equal to the total weight of selected items when traveling along the  $(i, i + 1)$ . Let  $p_i$  be a real variable equal to the difference of the total profit of selected items and their total transportation cost when delivering them to city  $i + 1$ . Let  $T_i \subseteq T$ ,  $1 \leq i \leq n$ , denote a set of possible segments to which velocity  $v_i$  of the vehicle may relate, i.e.  $T_i = \{\tau \in T : (v_\tau^{min} \in [v_i^{min}, v_i^{max}]) \vee (v_\tau^{max} \in [v_i^{min}, v_i^{max}])\}$ , where  $v_i^{max} = v_{max} - \nu \sum_{j=1}^i w_j^c$  is the maximal possible velocity that the vehicle can move along  $(i, i + 1)$  when packing in all compulsory items only, and  $v_i^{min} = v_{max} - \nu \cdot \min\left(\sum_{j=1}^i w_j^{max}, W\right)$  the minimum possible velocity along  $(i, i + 1)$  after having packed in all items available in cities  $1, \dots, i$ . Actually, we have  $v_i \in [v_i^{min}, v_i^{max}]$ . When  $v_i \in [v_\tau^{min}, v_\tau^{max}]$  for  $\tau \in T$ , any point in between endpoints of  $\tau$  is a weighted sum of them. Let  $B_i$  denote a set of all breakpoints that the linear segments of  $T_i$  have. Then the value of the real variable

---

**Algorithm 1: The Pre-processing Algorithm**


---

```

1 while true do
2   set flag ← false;
3   for each city i from 1 to n do
4     calculate  $\bar{w}_i^{max}$ ;
5   for each city i from n to 1 do
6     for each item  $e_{ik} \in M_i : \neg(\mu_{ik}^u \vee \mu_{ik}^c)$  do
7        $c_{max} \leftarrow 0$ ;  $c_{min} \leftarrow 0$ ;
8       initialize  $flag' \leftarrow false$ ;
9       for each city j from i to n do
10        for each item  $e_{jl} \in M_j : \neg\mu_{jl}^u \wedge ((i \neq j) \vee (k > l))$  do
11          if  $(\neg\mu_{jl}^c) \wedge (w_{ik} \geq w_{jl}) \wedge (p_{ik} - c_{min} \leq p'_{jl})$  then
12             $p'_{ik} \leftarrow p'_{jl} + c_{min}$ ;
13             $flag' \leftarrow true$ ; break;
14          if  $\mu_{jl}^c \wedge (w_{ik} \leq w_{jl}) \wedge (p_{ik} - c_{max} > p'_{jl})$  then
15             $\mu_{ik}^c \leftarrow true$ ;
16             $p'_{ik} \leftarrow p'_{jl} + c_{max}$ ;
17             $flag \leftarrow true$ ;
18             $flag' \leftarrow true$ ; break;
19          if  $flag'$  then break  $c_{min} \leftarrow c_{min} + Rd_j \left( \frac{1}{v_{max} - \nu(\bar{w}_j^c + w_{ik})} - \frac{1}{v_{max} - \nu\bar{w}_j^c} \right)$ ;
20           $c_{max} \leftarrow c_{max} + Rd_j \left( \frac{1}{v_{max} - \nu\bar{w}_j^{max}} - \frac{1}{v_{max} - \nu(\bar{w}_j^{max} - w_{ik})} \right)$ ;
21          if  $flag'$  then break  $p'_{ik} \leftarrow c_{max}$ ;
22          if  $c_{max} < p_{ik}$  then  $\mu_{ik}^c \leftarrow true$ ;  $flag \leftarrow true$ 
23 if  $\neg flag$  then break set  $flag \leftarrow false$ ;
24 for each city i from 1 to n do
25   calculate  $\bar{w}_i^c$ ;
26 for each city i from n to 1 do
27   for each item  $e_{ik} \in M_i : \neg(\mu_{ik}^u \vee \mu_{ik}^c)$  do
28      $c_{max} \leftarrow 0$ ;  $c_{min} \leftarrow 0$ ;
29     initialize  $flag' \leftarrow false$ ;
30     for each city j from i to n do
31       for each item  $e_{jl} \in M_j : \neg\mu_{jl}^c \wedge ((i \neq j) \vee (k > l))$  do
32         if  $(\neg\mu_{jl}^u) \wedge (w_{ik} \leq w_{jl}) \wedge (p_{ik} - c_{max} > p'_{jl})$  then
33            $p'_{ik} \leftarrow p'_{jl} + c_{max}$ ;
34            $flag' \leftarrow true$ ; break;
35         if  $\mu_{jl}^u \wedge (w_{ik} \geq w_{jl}) \wedge (p_{ik} - c_{min} \leq p'_{jl})$  then
36            $\mu_{ik}^u \leftarrow true$ ;
37            $p'_{ik} \leftarrow p'_{jl} + c_{min}$ ;
38            $flag \leftarrow true$ ;
39            $flag' \leftarrow true$ ; break;
40         if  $flag'$  then break  $c_{min} \leftarrow c_{min} + Rd_j \left( \frac{1}{v_{max} - \nu(\bar{w}_j^c + w_{ik})} - \frac{1}{v_{max} - \nu\bar{w}_j^c} \right)$ ;
41         if  $c_{min} \geq p_{ik}$  then
42            $\mu_{ik}^u \leftarrow true$ ;
43            $p'_{ik} \leftarrow c_{min}$ ;
44            $flag \leftarrow true$ ;
45            $flag' \leftarrow true$ ; break;
46          $c_{max} \leftarrow c_{max} + Rd_j \left( \frac{1}{v_{max} - \nu\bar{w}_j^{max}} - \frac{1}{v_{max} - \nu(\bar{w}_j^{max} - w_{ik})} \right)$ ;
47         if  $flag'$  then break  $p'_{ik} \leftarrow c_{min}$ ;
48 if  $\neg flag$  then break

```

---

$y_{ib} \in [0, 1]$  is a weight assigned to the breakpoint  $b \in B_i$ ,  $b \sim (v_b, t_b)$ . When the linear combination  $\sum_{b \in B_i} v_b y_{ib}$  under the constraint  $\sum_{b \in B_i} y_{ib} = 1$  equals  $v_i$ , the linear combination  $\sum_{b \in B_i} t_b y_{ib}$  overestimates  $t'_i$ . This underestimates the resulting profit minus the total transportation cost and gives a valid lower

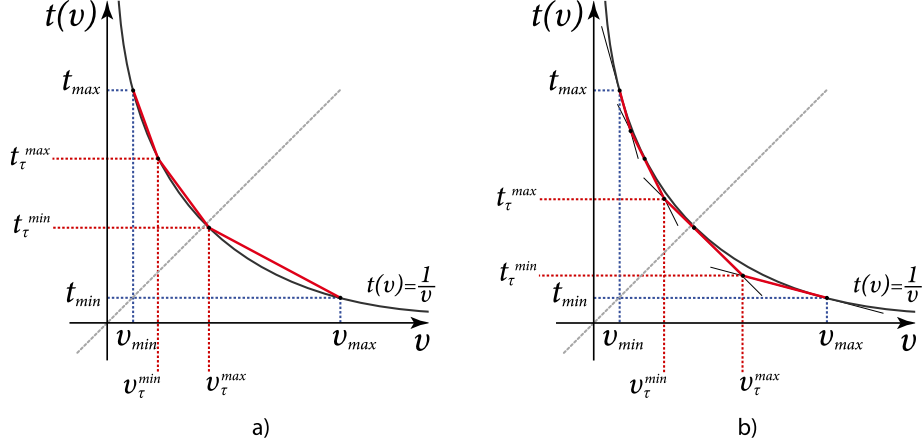


Figure 1: Piecewise linear approximation of  $t(v) = 1/v$

bound for  $\text{PWT}^c$  (and  $\text{PWT}^u$ ) that can be obtained by solving the following linear mixed 0-1 program ( $\text{LB}^\lambda$ ):

$$\max p^{LB}(x) = p_n \quad (5)$$

$$\text{s.t. } p_i = p_{i-1} + p_i^c + \sum_{e_{ik} \in M_i} p_{ik} x_{ik} - Rd_i \sum_{b \in B_i} t_b y_{ib}, \quad i = 1, \dots, n \quad (6)$$

$$w_i = w_{i-1} + w_i^c + \sum_{e_{ik} \in M_i} w_{ik} x_{ik}, \quad i = 1, \dots, n \quad (7)$$

$$\nu w_i + \sum_{b \in B_i} v_b y_{ib} = v_{max}, \quad i = 1, \dots, n \quad (8)$$

$$\sum_{b \in B_i} y_{ib} = 1, \quad i = 1, \dots, n \quad (9)$$

$$w_n \leq W \quad (10)$$

$$x_{ik} \in \{0, 1\}, \quad e_{ik} \in M \quad (11)$$

$$y_{ib} \in [0, 1], \quad i = 1, \dots, n, \quad b \in B_i \quad (12)$$

$$p_i \in \mathbb{R}, \quad i = 1, \dots, n \quad (13)$$

$$w_i \in \mathbb{R}_{\geq 0}, \quad i = 1, \dots, n \quad (14)$$

$$p_0 = w_0 = 0 \quad (15)$$

The value of  $\lambda$  in  $\text{LB}^\lambda$  sets its precision. Indeed, the precision of the lower bound may be increased at the cost of a running time as this also increases the number of segments, and thus raises the number of  $y$ -type variables to be involved. Equation (5) defines the objective function  $p^{LB}(x)$  as  $p_n$  that is the difference of the total profit of selected items delivered to city  $n+1$  and their total transportation cost. Since the transportation cost is approximated in  $\text{LB}^\lambda$ , the actual objective value for  $\text{PWT}^c$  (and  $\text{PWT}^u$ ) should be computed on the values of the decision variables of vector  $x$ . The resulting value then is also a valid lower bound. Equation (6) computes the difference  $p_i$  of the total profit

of selected items and their total transportation cost when arriving at city  $i + 1$  by summing up the value of  $p_{i-1}$  concerning  $(i - 1, i)$ , the profit of compulsory items  $p_i^c$  and the profit  $\sum_{e_{ik} \in M_i} p_{ik} x_{ik}$  of items selected in city  $i$ , and subtracting the approximated transportation cost along  $(i, i + 1)$ . Equation (7) gives the weight  $w_i$  of the selected items when the vehicle departs city  $i$  by summing up  $w_{i-1}$ , the weight of compulsory items  $w_i^c$  and the weight  $\sum_{e_{ik} \in M_i} w_{ik} x_{ik}$  of items selected in city  $i$ . Equation (8) implicitly defines segment  $\tau \in T_i$  to which the velocity of the vehicle  $v_i$  belongs and sets the weights for its endpoints. Equation (9) forces the total weight of the breakpoints of  $B_i$  be exactly 1. Equation (10) imposes the capacity constraint, and Eq. (11) declares  $x_{ik}$  as binary. Equation (12) states  $y_{ib}$  as a real variable defined in  $[0, 1]$ . Equation (13) declares  $p_i$  as a real variable, while Eq. (14) defines  $w_i$  as a non-negative real. Finally, Equation (15) establishes the base cases for  $p_0$  and  $w_0$ . Obviously, one can relax the integrality imposed on the  $x$ -type variables that leads to a linear programming model. In fact, this generally worsens the lower bound value, but gives an advantage in running time.

## 6.2. Upper Bound

We now describe the upper bound technique that adopts the piecewise linear approximation proposed for the lower bound. This time, our goal is to underestimate the traveling time  $t'_i$  that the vehicle spends to pass a distance unit when traveling along the  $(i, i + 1)$ , for any  $i = 1, \dots, n$ . We utilize the same set of breakpoints  $B_i$  generated from the set of linear segments  $T_i$ . In each point  $b \in B_i$ , we draw a tangent to the curve of the function  $t(v) = 1/v$  as depicted in Figure 1b. Subsequently, a new set of points  $\bar{B}_i$  is derived from the left and the rightmost points of  $B_i$ , and the points of intersection of each pair of neighboring tangents. This yields totally  $|B_i| + 1$  points that produce a new set of  $|B_i|$  linear segments  $\bar{T}_i$  resulted from connecting every two closest points in  $\bar{B}_i$ . Then a valid upper bound for  $\text{PWT}^c$  (and  $\text{PWT}^u$ ) can be obtained via the model of  $\text{LB}^\lambda$  with only the difference that the set of breakpoints  $\bar{B}_i$  is used instead of  $B_i$ . We designate this altered model as  $\text{UB}^\lambda$  and the corresponding objective function as  $p^{\text{UB}}(x)$ . Again, one can manage precision of the upper bound adjusting the value of  $\lambda$ . Furthermore, the integrality imposed on the  $x$ -type variables may be relaxed to speed up computations at the price of the upper bound's quality.

## 7. Mixed-Integer Programming-Based Approach

Both  $\text{PWT}^c$  and  $\text{PWT}^u$  belong to the specific class of fractional binary programming problems for which several efficient reformulation techniques exist to handle nonlinear terms. We follow the approach of Li (1994) and Tawarmalani et al. (2002) to reformulate  $\text{PWT}^c$  (and  $\text{PWT}^u$ ) as a linear mixed 0-1 program. It is applicable since the denominator of each fractional term in (1) is not equal to zero since  $v_{\min} > 0$ . We start with introduction of auxiliary real-valued variables  $y_i$ ,  $i = 1, \dots, n$ , such that  $y_i = 1 / \left( v_{\max} - \nu \sum_{j=1}^i \sum_{k=1}^{m_j} w_{jk} x_{jk} \right)$ . The

variables  $y_i$  express the travel time per distance unit along the edge  $(i, i + 1)$ . According to Li (1994), we can reformulate PWT<sup>c</sup> as a mixed 0-1 quadratic program by replacing (1) with (16) and adding the set of constraints (17) and (18).

$$\max \sum_{i=1}^n \left( \sum_{k=1}^{m_i} p_{ik} x_{ik} - R d_i y_i \right) \quad (16)$$

$$\text{s.t. } v_{max} y_i + \nu \sum_{j=1}^i \sum_{k=1}^{m_j} w_{jk} x_{jk} y_i = 1, \quad i = 1, \dots, n \quad (17)$$

$$y_i \in \mathbb{R}_+, \quad i = 1, \dots, n \quad (18)$$

According to Tawarmalani et al. (2002), if  $z = xy$  is a polynomial mixed 0-1 term where  $x$  is binary and  $y$  is a real variable, then it can be linearized via the set of linear inequalities: (i)  $z \leq Ux$ ; (ii)  $z \geq Lx$ ; (iii)  $z \leq y + L(x - 1)$ ; (iiii)  $z \geq y + U(x - 1)$ . Here,  $U$  and  $L$  are the upper and lower bounds on  $y$ , i.e.  $L \leq y \leq U$ . We can linearize the  $x_{jk} y_i$  term in (17) by introducing a new real variable  $z_{jk}^i = x_{jk} y_i$ . Let  $p_i^c$  and  $w_i^c$  denote the total profit and the total weight of the compulsory items in city  $i$  obtained in respect to Proposition 2. Similarly, let  $w_i^{max}$  be the total weight of the items (including all the compulsory items) in city  $i$ . Then variable  $y_i$ ,  $i = 1, \dots, n$ , can be bounded from below by  $L_i = 1 / (v_{max} - \nu \sum_{j=1}^i w_j^c)$  and from above by  $U_i = 1 / (v_{max} - \nu \cdot \min(\sum_{j=1}^i w_j^{max}, W))$ . In summary, we can formulate PWT<sup>c</sup> (and PWT<sup>u</sup>) as the following linear mixed 0-1 program (MIP<sup>λ</sup>):

$$\max p^{MIP}(x) = \sum_{i=1}^n \left( p_i^c + \sum_{k=1}^{m_i} p_{ik} x_{ik} - R d_i y_i \right)$$

$$\text{s.t. } v_{max} y_i + \nu \left( w_i^c + \sum_{j=1}^i \sum_{k=1}^{m_j} w_{jk} z_{jk}^i \right) = 1, \quad i = 1, \dots, n$$

$$z_{jk}^i \leq U_i x_{jk}, \quad i, j = 1, \dots, n, \quad j \leq i, \quad e_{jk} \in M_j$$

$$z_{jk}^i \geq L_i x_{jk}, \quad i, j = 1, \dots, n, \quad j \leq i, \quad e_{jk} \in M_j$$

$$z_{jk}^i \geq y_i + U_i (x_{jk} - 1), \quad i, j = 1, \dots, n, \quad j \leq i, \quad e_{jk} \in M_j$$

$$z_{jk}^i \leq y_i + L_i (x_{jk} - 1), \quad i, j = 1, \dots, n, \quad j \leq i, \quad e_{jk} \in M_j$$

$$\sum_{i=1}^n \sum_{k=1}^{m_i} w_{ik} x_{ik} \leq W \quad (19)$$

$$p^{LB}(x) \leq p^{MIP}(x) \leq p^{UB}(x) \quad (20)$$

$$x_{ik} \in \{0, 1\}, \quad e_{ik} \in M$$

$$z_{jk}^i \in \mathbb{R}_+, \quad i, j = 1, \dots, n, \quad j \leq i, \quad e_{jk} \in M_j$$

$$y_i \in \mathbb{R}_+, \quad i = 1, \dots, n$$

A solution of LB<sup>λ</sup> can be used as a starting solution for MIP<sup>λ</sup> and can yield the lower bound value  $p^{LB}(x)$ . In its turn, UB<sup>λ</sup> can provide the upper bound

value  $p^{UB}(x)$ . To set the value of  $\lambda$  to be used in both  $LB^\lambda$  and  $UB^\lambda$ , we specify its value through the notation  $MIP^\lambda$ . To strengthen the relaxation of  $MIP^\lambda$ , the sequencing constraints of Section 4 can be imposed as valid inequalities as has been earlier proposed in Polyakovskiy & Neumann (2015). However, our current investigations show that they are not beneficial anymore when the upper bound produced by  $UB^\lambda$  is applied in Eq. 20. An effective set of inequalities in order to obtain tighter relaxations can be obtained from the reformulation-linearization technique by Sherali & Adams (1999) who uses  $3n$  additional inequalities for the capacity constraint (19). Specifically, multiplying (19) by  $y_l$ ,  $U_l - y_l$  and  $y_l - L_l$ ,  $l = 1, \dots, n$ , we obtain the following inequalities:

$$\begin{aligned} \sum_{i=1}^n \sum_{k=1}^{m_i} w_{ik} z_{ik}^l &\leq W y_l; \\ U_l \sum_{i=1}^n \sum_{k=1}^{m_i} w_{ik} x_{ik} - \sum_{i=1}^n \sum_{k=1}^{m_i} w_{ik} z_{ik}^l &\leq U_l W - W y_l; \\ \sum_{i=1}^n \sum_{k=1}^{m_i} w_{ik} z_{ik}^l - L_l \sum_{i=1}^n \sum_{k=1}^{m_i} w_{ik} x_{ik} &\leq W y_l - L_l W. \end{aligned}$$

## 8. Branch-Infer-and-Bound Approach

Constraint programming (CP) has been shown to be a promising solution technique for various combinatorial optimization problems (Rossi et al. (2006, 2008)). It deals with a problem consisting of a set of variables  $X = \{x_1, \dots, x_n\}$  and a finite set of constraints  $C$  given on the elements of  $X$ . Each variable  $x_i \in X$  is associated with a domain  $D_i$  of available values. When the domain of every variable  $x_i$  is reduced to a singleton  $\{v_i\}$ , a values vector  $v = (v_1, \dots, v_n)$  is obtained. A satisfiability problem asks for a decision vector  $x = v$  such that all the constraints in  $C$  are satisfied simultaneously. A constraint optimization problem involves in addition an objective function  $f(x)$  that is to be either maximized or minimized over the set of all feasible solutions. In CP, constraints are given in a declarative way, but are viewed individually as special-purpose procedures that operate on a solution space. Each procedure applies a filtering algorithm that eliminates those values from the domains of the variables which cannot be a part of any feasible solution with respect to that constraint. The restricted domains generated by the constraints are in effect elementary in-domain constraints that restrict a variable to a domain of possible values. They become a part of a constraint store. To link all the procedures together in order to solve a problem as a whole, the constraint store is passed on to the next constraint to be processed. In such a way, the results of one filtering procedure are propagated to the others. In general, filtering algorithms are called repeatedly to achieve a certain level of consistency. This is because achieving arc consistency for one constraint might make other constraints inconsistent. Therefore, multiple runs of filtering are required for those constraints that share common variables of  $X$ . This process is called constraint propagation. CP aims to enumerate solutions

in respect to the constraint store in order to find the best feasible solution. To cope with this, a search tree is used, and every variable  $x_i$  with domain  $D_i$  is examined in some node of the tree. If  $D_i = \emptyset$ , an infeasible solution is found. If  $|D_i| > 1$ , one can branch on  $x_i$  by partitioning  $D_i$  into smaller domains, each corresponding to a branch. The domains of the variables decrease as they are reduced via constraint propagation when one descends into the tree. In the case of the constraint optimization problem, the search continues unless either the best solution is determined over those solutions where all the domains are singletons, or at least one of the domains is empty for every leaf node of the search tree. Certainly, the order in which the variables are instantiated and how the domains are partitioned matters for a running time.

Combining CP with the branch-and-cut method is a natural hybridization which results from the complementary strengths of both techniques. It gives rise to the so-called branch-infer-and-relax (BIR) approach presented by Bockmayr & Hooker (2005). The idea of BIR is to combine filtering and propagation used in CP with relaxation and cutting plane generation used in MIP. In each node of a search tree, constraint propagation creates a constraint store of i-domain constraints, while polyhedral relaxation creates a constraint store of inequalities. The two constraint stores can enrich each other, since reduced domains impose bounds on variables, and bounds on variables can reduce domains. The inequality relaxation is solved to obtain a bound on the optimal value, which prunes the search tree as in the branch-and-cut method.

Here, to solve  $\text{PWT}^c$  (and  $\text{PWT}^u$ ), we adopt this idea and introduce a branch-infer-and-bound approach that compounds CP and the upper bound introduced in Section 6.2. Specifically, we substitute the relaxation used in BIR with a stand-alone upper bound procedure to be executed in each node of the search tree in order to prune some of its branches. In each node, we create a refined set of items  $M' = M \setminus \cup_{e_{ik} \in M} e_{ik} : |D_{ik}| = 1$  and add the weight and profit of those items whose  $D_{ik} = \{1\}$  to  $w_i^c$  and  $w_i^{max}$  in the model of  $p^{UB}(x)$ , respectively. In other words, we treat the items accepted by the search as compulsory. Finally, we apply  $\text{UB}^\lambda$  to  $x$  formed on  $M'$  and prune a branch if the resulting  $p^{UB}(x)$  is smaller than the objective value of the best incumbent solution known.

Similarly to the previous MIP formulations, our CP model bases the search on binary decision vector  $x$  where variable  $x_{ik}$  takes the value of 1 to indicate that item  $e_{ik} \in M$  is chosen. To speed up computations, it employs an auxiliary integer variable  $w_i$  that calculates the total weight of the items selected in city  $i$  and all the preceding cities when traveling along the edge  $(i, i+1)$ , for any  $i = 1, \dots, n$ . Again,  $w_i^c$  and  $p_i^c$  denote the total weight and the total profit of compulsory items collected in city  $i$ . Here, we assume that the both values come out of the pre-processing step. The model has the following objective function and constraints ( $\text{BIB}^\lambda$ ):



$$\max p^{BIB}(x) = \sum_{i=1}^n \left( p_i^c + \sum_{k=1}^{m_i} p_{ik} x_{ik} - \frac{Rd_i}{v_{max} - \nu w_i} \right) \quad (21)$$

$$\text{s.t. } w_i = w_{i-1} + w_i^c + \sum_{e_{ik} \in M_i} w_{ik} x_{ik}, \quad i = 1, \dots, n \quad (22)$$

$$\sum_{i=1}^n \left( w_i^c + \sum_{k=1}^{m_i} w_{ik} x_{ik} \right) \leq W \quad (23)$$

$$x_{il} \leq x_{ik}, \quad i = 1, \dots, n, \quad e_{il}, e_{ik} \in M_i : (p_{il} < p_{ik}) \wedge (w_{il} \geq w_{ik}), \quad l \neq k, \quad (24)$$

$$x_{jl} \leq x_{ik}, \quad i = 1, \dots, n, \quad j < i, \quad e_{jl} \in M_j, \quad e_{ik} \in M_i : (p_{jl} - \Delta_l^{ji} < p_{ik}) \wedge (w_{jl} \geq w_{ik}) \quad (25)$$

$$x_{jl} \geq x_{ik}, \quad i = 1, \dots, n, \quad j < i, \quad e_{jl} \in M_j, \quad e_{ik} \in M_i : (p_{jl} - \overline{\Delta}_l^{ji} > p_{ik}) \wedge (w_{jl} \leq w_{ik}) \quad (26)$$

$$\text{sequencing}(x_{jl}, [w_1, \dots, w_n]), \quad j = 1, \dots, n,$$

$$e_{jl} \in M_j : (\exists e_{ik} \in M_i, j < i : p_{jl} - \overline{\Delta}_l^{ji} \leq p_{ik} \leq p_{jl} - \Delta_l^{ji}) \quad (27)$$

$$x_{ik} \in \{0, 1\}, \quad e_{ik} \in M \quad (28)$$

$$w_i \in \{0, \dots, W\}, \quad i = 1, \dots, n \quad (29)$$

$$w_0 = 0 \quad (30)$$

Function 21 represents the objective function of the problem. For each edge  $(i, i+1)$ ,  $i = 1, \dots, n$ , it sums up the profits of items taken in city  $i$  minus the cost of transportation of all the items that have been placed to the vehicle in city  $i$  and all the cities prior to  $i$ . Equation (22) calculates the weight  $w_i$  of all the items taken in the cities  $1, \dots, i$ . Equation (23) is a capacity constraint. Equations (25), (26) and (27) impose the set of redundant sequencing constraints  $SC^i$ ,  $SC^{ii}$ , and  $SC^{iii}$  of Section 4, respectively. This set may be rather small, and therefore might have a limited impact on inference of in-domain constraints during the search. Indeed, more constraints might be involved if  $w_b^c$  in  $\Delta_l^{ji}$  of constraint  $SC^{ii}$  was larger and  $w_b^{max}$  in  $\overline{\Delta}_l^{ji}$  of constraint  $SC^{iii}$  was smaller. These two variables  $w_b^c$  and  $w_b^{max}$  can in fact be considered as initial lower and upper bounds on the weight of the items collected in city  $b$ . As one descends into the tree, items are either collected or rejected. Being picked up in some city, an item contributes its weight that increases the lower bound. Being rejected, it lowers the upper bound. We include those constraints into the pool that still may work out when the lower or upper bound on the weight reaches a certain level. Specifically, (27) adds a redundant customized constraint for each item  $e_{jl}$  that has at least one related  $e_{ik}$ ,  $j < i$ , such that their mutual sequencing depends on the weight of the items collected in cities  $j, \dots, i-1$ . Equations (28) and (29) define the domains of the variables. Finally, Equation (30) sets the base case  $w_0 = 0$ .

We assume a depth-first search strategy for traversing the binary search tree and instantiate variables in the order in which the cities appear in  $N$ . No order is given to the items within the same city. Therefore, at the moment when item  $e_{ik}$  is to be instantiated by the search, the domains of the variables associated with the items in cities  $1, \dots, i-1$  and with those in city  $i$  that appear prior

to  $e_{ik}$  in the set  $M_i$  have been already reduced to singletons. Accordingly, the domain of variable  $w_i$  is reduced to a singleton once the decision variables of the items in city  $i$  have been all fixed to singletons.

The customized sequencing constraint **sequencing**( $x_{jl}, [w_1, \dots, w_n]$ ) applies to variable  $x_{jl}$  for which there exists at least one item, say  $x_{ik}$ , such that  $p_{jl} - \bar{\Delta}_l^{ji} \leq p_{ik} \leq p_{jl} - \Delta_l^{ji}$  holds and  $j < i$ . Algorithm 2 sketches the pseudocode of the corresponding filtering algorithm. We use a Boolean variable  $\Theta_{e_{jl}e_{ik}}$ , which takes value *true* to indicate the situation when selection of items  $e_{jl}$  and  $e_{ik}$  may be potentially sequenced. First, the algorithm initializes variables  $\bar{w}^{max}$  and  $\bar{w}^{min}$  that, respectively, represent the upper and lower bounds on the weight of collected items that the vehicle has in city  $i$ . It sets both variables to the sum of weights of items collected in the cities prior to city  $j$ , i.e.  $w_{j-1}$ , and adds the weight of item  $e_{jl}$  if its variable  $x_{jl}$  has been fixed to 1 (cf. line 1). Then the algorithm starts exploring the items positioned in the cities succeeding  $j$ . Each time the next city  $i$  is taken into consideration, it adds the weight of all the items existing in  $i$  to  $\bar{w}^{max}$  and the weight of all the compulsory items in  $i$  to  $\bar{w}^{min}$  (cf. line 5). In each city  $i$ , the algorithm examines the items that the city contains. When the corresponding variable  $x_{ik}$  of item  $e_{ik}$  is a singleton, the algorithm modifies the bounds on collected weight accordingly (cf. lines 10 and 10). Subsequently, if  $\Theta_{e_{jl}e_{ik}}$  is *true*, it tries to establish a sequencing relation between  $e_{ik}$  and  $e_{jl}$ . If the condition in line 12 holds, it excludes 1 from domain  $D_{ik}$ , and therefore declines item  $e_{ik}$  since  $e_{jl}$  is assumed to be rejected by the search. At the same time, it lowers  $\bar{w}^{max}$  by the value of  $w_{ik}$ . On the other hand, if the condition in line 17 holds, it excludes 0 from domain  $D_{ik}$ , and therefore selects item  $e_{ik}$  because  $e_{jl}$  is assumed to be selected. In addition, it increases  $\bar{w}^{min}$  by the value of  $w_{ik}$ . If no relation has been established at that stage, the algorithm adds  $e_{ik}$  to the set of items  $M^*$ . Continuing to explore other items, it tries to prove each item of  $M^*$  to be unprofitable every time it finishes investigating city  $i$  (cf. line 30). Furthermore, having all the cities analyzed, the algorithm returns back to set  $M^*$  and tries to prove each of its items to be compulsory if the instance  $I$  at hands refers to PWT<sup>u</sup> (cf. line 36).

## 9. Computational Experiments

In this section, we investigate the effectiveness of the proposed approaches by experimental studies. On the one hand, we assess the advantage of the pre-processing scheme in terms of quantity of discarded items and auxiliary decision variables. On the other hand, we evaluate our  $LB^\lambda$ ,  $MIP^\lambda$ , and  $BIB^\lambda$  models in terms of solution quality and running time. The program code is implemented in JAVA using the IBM OPTIMIZATION STUDIO 12.6.2. To solve the mixed-integer programs  $LB^\lambda$ ,  $UB^\lambda$  and  $MIP^\lambda$ , we use CPLEX with default settings. When running  $UB^\lambda$ , we increase CPLEX's precision by setting the relative tolerance on the gap between the best integer objective and the objective of the best node remaining to  $1e-7$ . To solve the constrained program within  $BIB^\lambda$ , we use CP OPTIMIZER switched to the depth-first search mode and set the relative

---

**Algorithm 2:** The Filtering Algorithm of Constraint sequencing( $x_{jl}, [w_1, \dots, w_n]$ )

---

```

1 initialize  $\overline{w}^{max} \leftarrow w_{j-1} + w_{jl}x_{jl}$ ; initialize  $\overline{w}^{min} \leftarrow w_{j-1} + w_{jl}x_{jl}$ ;
2 initialize  $M^* \leftarrow \emptyset$ ;
3 set  $\overline{\Delta}_l^{jj} \leftarrow 0$ ;  $\Delta_l^{jj} \leftarrow 0$ ;
4 for each city  $i$  from  $j$  to  $n$  do
5    $\overline{w}^{max} \leftarrow \overline{w}^{max} + w_i^{max}$ ;  $\overline{w}^{min} \leftarrow \overline{w}^{min} + w_i^c$ ;
6   for each item  $e_{ik} \in M_i, e_{ik} \neq e_{jl}$  do
7     initialize  $flag \leftarrow true$ ;
8     if  $DomainSize(D_{ik}) \leq 1$  then
9        $flag \leftarrow false$ ;
10    if  $x_{ik} = 0$  then  $\overline{w}^{max} \leftarrow \overline{w}^{max} - w_{ik}$  if  $x_{ik} = 1$  then  $\overline{w}^{min} \leftarrow \overline{w}^{min} + w_{ik}$ 
11    if  $\Theta_{e_{jl}e_{ik}}$  then
12      if  $(w_{jl} \leq w_{ik}) \wedge (p_{jl} - \overline{\Delta}_l^{ji} > p_{ik}) \wedge (x_{jl} = 0)$  then
13        if  $DomainSize(D_{ik}) = 2$  then
14           $\overline{w}^{max} \leftarrow \overline{w}^{max} - w_{ik}$ ;
15           $flag \leftarrow false$ ;
16          RemoveValue( $D_{ik}, 1$ );
17        if  $(w_{jl} \geq w_{ik}) \wedge (p_{jl} - \Delta_l^{ji} < p_{ik}) \wedge (x_{jl} = 1)$  then
18          if  $DomainSize(D_{ik}) = 2$  then
19             $\overline{w}^{min} \leftarrow \overline{w}^{min} + w_{ik}$ ;
20             $flag \leftarrow false$ ;
21            RemoveValue( $D_{ik}, 0$ );
22    if  $flag$  then
23       $M^* \leftarrow M^* \cup \{e_{ik}\}$ ;
24      initialize  $\overline{\Delta}_k^{in+1} \leftarrow 0$ ;  $\Delta_k^{in+1} \leftarrow 0$ ;
25   $\overline{\Delta}_l^{ji+1} \leftarrow \overline{\Delta}_l^{ji} + Rd_i \left( \frac{1}{v_{max} - \nu \min(\overline{w}^{max}, W)} - \frac{1}{v_{max} - \nu (\min(\overline{w}^{max}, W) - w_{ik})} \right)$ ;
26   $\Delta_l^{ji+1} \leftarrow \Delta_l^{ji} + Rd_j \left( \frac{1}{v_{max} - \nu (\min(\overline{w}^{min}, W) + w_{ik})} - \frac{1}{v_{max} - \nu \min(\overline{w}^{min}, W)} \right)$ ;
27  for each item  $e_{ab} \in M^*$  do
28     $\overline{\Delta}_b^{an+1} \leftarrow \overline{\Delta}_b^{an+1} + Rd_i \left( \frac{1}{v_{max} - \nu \min(\overline{w}_j^{max}, W)} - \frac{1}{v_{max} - \nu (\min(\overline{w}^{max}, W) - w_{ab})} \right)$ ;
29     $\Delta_b^{an+1} \leftarrow \Delta_b^{an+1} + Rd_j \left( \frac{1}{v_{max} - \nu (\min(\overline{w}^{min}, W) + w_{ab})} - \frac{1}{v_{max} - \nu \min(\overline{w}^{min}, W)} \right)$ ;
30    if  $p_{ab} - \Delta_b^{an+1} \leq 0$  then
31      RemoveValue( $D_{ab}, 1$ );
32       $\overline{w}^{max} \leftarrow \overline{w}^{max} - w_{ab}$ ;
33       $M^* \leftarrow M^* \setminus \{e_{ab}\}$ ;
34 if  $ProblemType(I) \equiv PWT^u$  then
35   for each item  $e_{ab} \in M^*$  do
36     if  $p_{ab} - \overline{\Delta}_b^{an+1} > 0$  then RemoveValue( $D_{ab}, 0$ )

```

---

tolerance gap to 0. Furthermore, we limit the parallel mode to only a single thread for CPLEX and CP OPTIMIZER to make them both comparable with respect to each other when dealing with the small size instances of Section 9.1. We set the number of threads to the maximum number of cores available when investigating the large size instances in Section 9.2.

The test instances are adopted from the benchmark set  $B$  of Polyakovskiy et al. (2014). This benchmark set is constructed on TSP instances from TSPLIB introduced by Reinelt (1991) augmented by a set of items distributed among all the cities but the first one. We use the set of items available in each city and obtain the route from the corresponding TSP instance by running the Chained Lin-Kernighan heuristic proposed by Applegate et al. (2003). Given the permutation  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$  of the cities computed by the Chained Lin-Kernighan

heuristic, where  $\pi_1$  is free of items, we use  $N = (\pi_2, \pi_3, \dots, \pi_n, \pi_1)$  as the route for our problem. We consider the *uncorrelated* (uncorr), *uncorrelated with similar weights* (uncorr-s-w), and *bounded strongly correlated* (b-s-corr) types of items' generation, and set  $v_{min}$  and  $v_{max}$  to 0.1 and 1 as proposed for  $B$ .

### 9.1. Computational Experiments on the Set of Small Size Instances

First, we investigate three families of small size instances based on the TSP problems **eil51**, **eil76**, and **eil101** with 51, 76 and 101 cities, respectively. This series of experiments has been carried out on PC with 4 Gb RAM and a 3.06 GHz Dual Core processor. The results of the experiments are shown in Table 1. All the instances of a family have the same route  $N$ . We consider instances with 1, 5, and 10 items per city. The postfixes 1, 6 and 10 in the instances' names indicate the vehicle's capacity  $W$ . The greater the value of a postfix is, the larger  $W$  is given. Column 2 specifies the total number of items  $m$ . A ratio  $\alpha = 100(m - m')/m$  in Column 3 denotes a percentage of items discarded in pre-processing step, where  $m'$  is the number of items left after pre-processing. Column *ver* identifies by "u" whether  $PWT^c$  has been reduced to  $PWT^u$  by pre-processing. Columns 5-7 report results for  $LB^\lambda$  with  $\lambda = 100$ . Specifically, column 5 gives  $\rho$  as a ratio between the lower bound obtained by  $LB^{100}$  and the optimum obtained by the branch-infer-and-bound approach. Column 6 contains the running time  $t$  of  $LB^{100}$ . Column 7 shows a rate  $\beta$  that is a percentage of auxiliary  $y$ -type variables used in practice by  $LB^\lambda$ . At most  $\lambda n$  variables is required by  $LB^\lambda$ . Thus,  $\beta$  is computed as  $\beta = 100(\sum_{i=1}^n |B_i|) / (\lambda n)$ . Column 8 presents the running time  $t$  for the MIP-based exact approach  $MIP^\lambda$  when  $\lambda$  is set to 1000. Therefore, both  $LB^\lambda$  and  $UB^\lambda$  incorporated into  $MIP^\lambda$  use  $\lambda = 1000$  to compute initial lower and upper bounds. The time limit of 1 day has been given to  $MIP^\lambda$  in total, while  $LB^\lambda$  and  $UB^\lambda$  got the time limit of 2 hours each. The rest columns of the table describe results for the branch-infer-and-bound approach  $BIB^\lambda$  with  $\lambda \in \{500, 1000, 1500\}$ . Furthermore, two cases of  $BIB^\lambda$  for  $\lambda = 1000$  have been studied.  $BIB^{1000}$  is exactly that one which is described in Section 8.  $BIB_{no\ seq.}^{1000}$  is its copy that does not include the customized sequencing constraint to the model. In such way, we evaluate impact of the constraint on the performance of  $BIB^\lambda$ . We employ  $LB^{100}$ , which we give 2 hours of running time limit, to provide  $BIB^\lambda$  with a lower bound to be used then to prune the search tree. Within  $BIB^{500}$ ,  $BIB^{1000}$ ,  $BIB_{no\ seq.}^{1000}$ , and  $BIB^{1500}$ , we run the variant of  $UB^\lambda$  where the integrality constraints on  $x$ -type decision variables are removed. This makes  $UB^\lambda$  a linear program and significantly speeds up computations at the very small cost of solution quality. Each of the columns  $t$  reports computational time for the corresponding  $BIB^\lambda$ . Similarly each of the columns named  $\omega$  does when presenting  $\omega$  as a relative gap in percents that compares the running time of  $BIB^\lambda$  to that one taken by  $MIP^{1000}$ . When the running time of  $MIP^{1000}$ , say  $t^{MIP}$ , is less than the running time of  $BIB^\lambda$ , say  $t^{BIB}$ ,  $\omega$  is computed as  $\omega = 100(t^{BIB} - t^{MIP})/t^{MIP}$ . Otherwise,  $\omega = -100(t^{MIP} - t^{BIB})/t^{BIB}$ . Therefore, the positive values of  $\omega$  evaluate superiority of  $MIP^\lambda$  against  $BIB^\lambda$ . And vice verse, negative values

reveal that  $\text{BIB}^\lambda$  gives an advantage. The entries of the table marked by “-” indicate that results have not been obtained within the given time limit. The least running time obtained for a particular instance is marked by bold and underlined in the entry  $t$  of the corresponding approach.

Table 2 presents the details on the constraint programming search performed by  $\text{BIB}^\lambda$  on small size instances of the benchmark suite. Columns 1-3 specify the instance’s name, the total number of items  $m$ , and the version of the problem solved. Other columns of the table describe results for  $\text{BIB}^\lambda$  with  $\lambda \in \{500, 1000, 1500\}$ . Each of the sections shows the number of branches  $b$  totally explored and the number of fails  $f$  obtained by the CP solver during the search when the corresponding parameter value  $\lambda$  is in use. Furthermore, each of the columns “ $\text{UB}^\lambda$  runs” reports the number of successful  $\text{UB}^\lambda$  runs, say  $r^s$ , that resulted in pruning of the search tree and the total number of runs, say  $r^t$ . The two values are separate by “|”. In the parentheses, it also gives a percentage of successful runs  $\eta$  computed as  $\eta = 100r^s/r^t$ .

The results of the experiments demonstrate efficiency of the pre-processing scheme. It is rather strong with respect to the instances of uncorr type and removes on average 31.6% of their items. Concerning the uncorr-s-w type of the instances, it is able to exclude on average 18.5% of the items that they contain. Within these two categories, the instances with large  $W$  are rather liable to reduction. Because  $W$  is large, they get more chances to loose enough items so that the total weight of rest items becomes less or equal to  $W$ . The pre-processing scheme does not work well for the b-s-corr type of the instances. No instance of this type has been reduced to  $\text{PWT}^u$ . Because the profit of an item approaches its weight, the pre-processing encounters a difficulty to find unprofitable items for this instance type. In general, the way of how profits and weights are generated is not an obstacle in itself for the pre-processing to be successful. There are other factors, like the value of renting rate  $R$ , the value of capacity  $W$ , and a distance to the destination from the city where an item is positioned, that hinder its application. For example, if a route is long enough, some items in the first cities can be shown to be unprofitable even in the case of their b-s-corr type of generation. Clearly, the fact that we cannot handle the instances of this type is a proper property of the benchmark suite  $B$ .

$\text{LB}^{100}$  is particularly fast and its model is solved to optimality in a very short time for all the small size instances. Only one instance of the whole test suite causes a difficulty in terms of a running time. The approximate approach looks very swift even with instances of the b-s-corr type and produces very good approximation for reasonably small  $\lambda = 100$ . The ratio  $\rho$  close to 1 points out that  $\text{LB}^{100}$  obtains approximately the same result as the optimal objective value is, but in a shorter time. Therefore,  $\text{LB}^\lambda$  gives an advanced trade-off in terms of computational time and solution’s quality comparing to the exact approaches. The larger  $\lambda = 1000$  has been tested in our earlier experiments (Polyakovskiy & Neumann (2015)). However, it results to a very limited improvement in the value of the total reward at the larger cost of a running time, and more importantly at the large cost of a memory usage. Indeed, as  $\lambda$  increases, the approach requires more memory as the number of auxiliary variables grows. The

rate  $\beta$  demonstrates that in practice  $LB^\lambda$  uses a very reduced set of auxiliary variables. The average over all the entries is just 48.5%. Therefore, less than a half of all possible variables is used only. In general,  $\beta$  is significantly small when  $W$  is large, since latter results in a slower growth of diapason  $[v_i^{\min}, v_i^{\max}]$  in  $LB^\lambda$ , for  $i = 1, \dots, n$ . In other words, the instances with large  $W$  require less number of auxiliary decision variables comparing to the instances where  $W$  is smaller.

Comparing to the earlier results, we see now that the unconstrained instances of the problem are not to be easier to handle as it has been previously observed (Polyakovskiy et al. (2014); Polyakovskiy & Neumann (2015)). Now, our mixed-integer programming approach is able to solve all the small size instances to optimality as it is strengthened with the upper bound  $UB^\lambda$ . Specifically,  $MIP^\lambda$  takes much less time than the given limit and can be executed on an ordinary PC instead of the highly productive computational cluster that has been previously utilized. The results show that the instances of the uncorr-s-w type are harder to solve for  $MIP^\lambda$  comparing to other types. This fact looks interesting. Comparing to the b-s-corr type, this type has around 20% of items per instance excluded by the pre-processing step and needs less 50% of auxiliary variables, but takes more time to reach a solution. This also differs PWT from the classical 0-1 knapsack problem for which the b-s-corr type is shown to be the hardest one (Martello et al. (1999)). The b-s-corr type is not easier to solve than the uncorr type in terms of running time, nor the latter is when compared to the former.  $MIP^\lambda$  outperforms  $BIB^\lambda$  mainly on the set of b-s-corr type instances with the least capacity  $W$ .  $BIB^{500}$  is superior on the wide range of instances. It is highly effective for the set of uncorr and uncorr-s-w type instances with large capacity. Depending on the parameter  $\lambda$ , performance of  $BIB^\lambda$  can be further improved for some instances. Specifically, setting  $\lambda$  to larger values allows  $BIB^\lambda$  to solve b-s-corr type instances with many items and large capacity at a faster speed. The further increase of  $\lambda$  up to 1500, leads to the best result for some of those instances. However, this degrades performance of  $BIB^\lambda$  on other instances. Performance of  $BIB^\lambda$  with the values of  $\lambda$  less than 500 and greater than 1500 have also been investigated. However, using too small or too large values increases the running time of the approach. The same behavior has been observed for  $MIP^\lambda$ , for which the value of 1000 is the most promising one. In summary, we argue that selecting  $\lambda = 1000$  represents a good balance as  $BIB^{1000}$  works reasonably fast over all the instances.

The sequencing constraint of  $BIB^\lambda$  is quite weak when dealing with the instances of the b-s-corr type. Therefore,  $BIB^\lambda$  needs to tighten the upper bound to reach a better performance. The details on the CP Search reported in Table 2 show that the number of  $UB^\lambda$  runs considerably decreases when  $\lambda$  increases. Considering other types, the growth of the number of runs is not that much for them. Moreover, the number of explored branches  $b$  and the number of fails  $f$  obtained stay almost the same for different values of  $\lambda$ . This means that  $BIB^\lambda$  extensively rely on the sequencing constraint when solving the uncorr and uncorr-s-w types of the instances. When we turn off the sequencing

constraint in the case of  $\text{BIB}_{no\ seq.}^{1000}$ , the approach explores more branches with respect to the uncorr and uncorr-s-w types and spends more time for this while the number of branches traversed and the running time stay almost the same for the b-s-corr type.

## 9.2. Computational Experiments on the Set of Large Size Instances

Solving large size instances is costly and requires computational capacity beyond that of an ordinary computer. Therefore, this series of experiments has been carried out on a computational cluster with 128 Gb RAM and 2.8 GHz 48-cores AMD Opteron processor. The goal of our second experiment is to understand how fast  $\text{LB}^\lambda$  handles instances of larger size and how efficient the pre-processing scheme is. We use the same settings for MIP solver as for our first experiment and set  $\tau = 100$ . We investigate two families of the largest size instances of benchmark suite  $B$ , namely those based on the TSP problems **pla33810** and **pla85900** with 33810 and 85900 cities, respectively. Table 3 reports the results. Columns 1-4 specify the instance's name, the total number of items  $m$ , the percentage of items discarded within the pre-processing step, and the version of the problem solved. Columns 5-7 contain, respectively, the running time  $t$  of  $\text{LB}^{100}$  (including the time taken by pre-processing), the percentage of auxiliary  $y$ -type variables used by it, and the pre-processing time spent. Columns 8 and 9 cover the case when  $\text{LB}^{100}$  is run without any pre-processing at all. Specifically, in a similar way as in Section 9.1,  $\omega$  represents a relative gap in percents that compares the running time of  $\text{LB}^{100}$  to that one taken by  $\text{LB}_{no\ pre-pr.}^{100}$ . Then  $\gamma$  is a ratio between the number of  $y$ -type variables used by  $\text{LB}_{no\ pre-pr.}^{100}$  and  $\text{LB}^{100}$ . Finally, the last column corresponds to the case when  $\text{LB}^{100}$  does not use the reasoning based on the sequencing constraints to accelerate deduction of compulsory and unprofitable items when a problem in hand is unconstrained (see Section 5 for details). Remind that in this situation the pre-processing scheme determines the properties of an item independently of other items whose properties are already known.

$\text{LB}^\lambda$  proves its ability to master large instances in a reasonable time. It needs less than  $\sim 30$  minutes to solve any instance of family **pla33810**. Almost all the instances of family **pla85900** can be solved within 1 hour; it takes no longer than  $\sim 4$  hours for any of them. The pre-processing scheme excludes on average 27.7% of items per instance of the uncorr type and 13.2% of items per instance of the uncorr-s-w type. These both types are vulnerable for reduction to  $\text{PWT}^u$  when their capacity values  $W$  are large. Furthermore, they are solved relatively fast as require much less number of auxiliary variables. The results show that the pre-processing step is importation and leads to great speeding-up. It accelerates computations by 322% on average for the uncorr type and by 422% on average for the uncorr-s-w type. In addition, the average value of  $\gamma$  is 1.5 for the uncorr type against 1.2 for the uncorr-s-w type. Interestingly, despite on the larger portion of auxiliary  $y$ -type variables excluded for the uncorr type, its speeding-up indicator  $\omega$  is less than that one of the uncorr-s-w type. Pre-processing is rather costly when applied to the unconstrained instances. Most of the times, a

Table 1: Results of Computational Experiments on Small Size Instances

instance	m	$\alpha$ , %	ver	LB <sup>100</sup>			MIP <sup>1000</sup>		BIB <sup>500</sup>		BIB <sup>1000</sup>		BIB <sup>1000</sup> <sub>no seq.</sub>		BIB <sup>1500</sup>		
				$\rho$	t, sec	$\beta$ , %	t, sec	t, sec	$\omega$ , %	t, sec	$\omega$ , %	t, sec	$\omega$ , %	t, sec	$\omega$ , %		
instance family e1151																	
uncorr_01	50	42.0	c	1.00000	0.2	55.8	2.3		1.7	-34.9	3.5	53.7	6.4	180.2	6.1	164.5	
uncorr_06	50	14.0	c	1.00000	0.2	39.2	3.2		0.9	-255.5	1.7	-94.5	3.2	-1.0	2.6	-24.9	
uncorr_10	50	12.0	u	1.00000	0.1	11.1	0.8		0.4	-104.2	0.5	-56.0	0.6	-35.9	0.6	-29.2	
uncorr-s-w_01	50	30.0	c	1.00000	0.3	77.4	2.9		1.6	-84.3	3.1	5.3	7.6	161.3	4.8	65.5	
uncorr-s-w_06	50	24.0	c	1.00000	0.1	35.8	1.4		0.9	-52.6	1.7	21.1	2.3	60.4	2.7	90.6	
uncorr-s-w_10	50	34.0	u	1.00000	0.1	13.2	1.6		0.3	-379.2	0.4	-337.1	0.5	-252.2	0.4	-274.2	
b-s-corr_01	50	4.0	c	1.00000	0.3	89.7	4.5	23.2	412.4	49.0	982.1	52.0	1049.0	77.8	1620.7		
b-s-corr_06	50	0.0	c	1.00000	0.2	53.4	2.5	2.9	14.0	6.4	152.1	6.3	146.0	11.0	333.1		
b-s-corr_10	50	0.0	c	1.00000	0.2	25.7	1.9	1.9	0.9	3.7	100.1	3.5	89.3	6.4	241.0		
uncorr_01	250	39.2	c	1.00000	0.3	65.5	10.4		4.6	-127.5	9.1	-15.1	21.5	106.0	14.7	41.0	
uncorr_06	250	16.4	c	1.00000	0.2	38.3	65.1		2.7	-2353.2	4.2	-1462.6	7.3	-791.5	6.1	-958.8	
uncorr_10	250	54.4	u	1.00000	0.1	10.9	20.4		0.8	-2512.1	1.0	-1939.4	1.8	-1029.1	1.2	-1532.5	
uncorr-s-w_01	250	20.8	c	1.00000	0.3	88.0	7.1		3.6	-97.4	6.8	-5.0	40.8	470.9	10.4	45.4	
uncorr-s-w_06	250	14.0	c	1.00000	0.2	44.6	41.7		1.6	-2520.9	2.4	-1647.6	5.8	-624.0	3.5	-1097.9	
uncorr-s-w_10	250	19.2	u	0.99998	0.2	15.7	83.3		1.1	-7761.7	1.3	-6492.9	2.3	-3473.2	1.5	-5407.9	
b-s-corr_01	250	0.0	c	1.00000	0.3	90.2	8.9	28.9	223.5	58.3	552.7	65.6	634.2	94.1	953.0		
b-s-corr_06	250	0.0	c	0.99997	0.2	55.8	20.9	27.3	31.0	53.7	157.3	55.4	165.7	57.4	175.0		
b-s-corr_10	250	0.0	c	1.00000	0.2	26.7	55.6		7.6	-633.3	9.8	-466.6	9.5	-484.3	16.1	-245.0	
uncorr_01	500	37.0	c	1.00000	0.3	67.7	12.9		11.5	-12.0	22.1	70.8	68.5	429.4	35.8	177.1	
uncorr_06	500	15.2	c	0.99993	0.3	38.8	81.6		5.6	-1369.8	8.3	-885.3	17.8	-357.3	12.0	-578.6	
uncorr_10	500	51.4	u	1.00000	0.2	11.6	93.1		1.3	-7093.6	1.5	-5924.9	3.4	-2664.6	1.9	-4915.4	
uncorr-s-w_01	500	20.2	c	1.00000	0.2	89.1	12.1		4.9	-149.1	8.7	-39.0	63.3	422.1	13.1	8.0	
uncorr-s-w_06	500	15.2	c	0.99990	0.2	44.2	147.7		3.0	-4854.8	3.9	-3640.9	9.9	-1394.9	5.3	-2675.1	
uncorr-s-w_10	500	18.6	u	1.00000	0.2	16.1	208.2		2.1	-9788.2	2.3	-8799.4	4.6	-4422.6	2.6	-8041.0	
b-s-corr_01	500	0.0	c	0.99993	0.3	91.3	29.9	226.7	657.2	324.1	982.6	396.9	1225.8	535.0	1687.3		
b-s-corr_06	500	0.0	c	0.99995	0.3	55.4	71.3	316.7	344.1	149.0	109.0	161.5	126.4	235.7	230.4		
b-s-corr_10	500	0.0	c	1.00000	0.2	26.0	100.8	87.9	-14.7	25.7	-292.7	25.6	-294.4	27.4	-268.5		
instance family e1176																	
uncorr_01	75	26.7	c	1.00000	0.3	76.7	5.4		5.0	-9.5	10.4	91.9	20.7	280.6	17.7	226.7	
uncorr_06	75	14.7	c	1.00000	0.3	33.9	9.8		1.8	-433.3	3.3	-195.4	4.3	-127.2	5.2	-88.3	
uncorr_10	75	48.0	u	1.00000	0.1	11.3	1.9		0.5	-303.0	0.6	-235.6	1.0	-89.9	0.7	-163.0	
uncorr-s-w_01	75	26.7	c	1.00000	0.4	78.2	4.9		4.8	-2.3	9.6	94.6	30.4	515.8	15.6	217.1	
uncorr-s-w_06	75	17.3	c	1.00000	0.3	40.7	7.8		1.3	-495.3	2.2	-254.7	5.0	-55.1	3.2	-140.3	
uncorr-s-w_10	75	16.0	u	1.00000	0.2	16.6	10.8		0.7	-1437.9	0.9	-1073.5	1.5	-600.8	1.1	-862.2	
b-s-corr_01	75	0.0	c	1.00000	0.4	93.5	6.2	215.6	3371.7	463.3	7362.1	510.3	8119.1	770.3	12305.6		
b-s-corr_06	75	0.0	c	1.00000	0.3	58.9	8.6		5.1	-68.3	11.1	29.3	11.5	33.5	19.1	121.6	
b-s-corr_10	75	0.0	c	1.00000	0.3	25.5	6.7		5.8	-15.5	9.8	46.1	10.1	50.5	11.1	64.8	
uncorr_01	375	38.1	c	1.00000	0.3	66.3	30.6		10.0	-204.5	19.9	-53.8	48.9	59.8	32.4	5.8	
uncorr_06	375	16.0	c	1.00000	0.2	37.0	162.0		5.5	-2825.0	9.6	-1591.1	17.5	-825.5	14.5	-1014.0	
uncorr_10	375	9.9	u	1.00000	0.2	11.8	105.4		1.4	-7412.1	1.6	-6400.6	6.0	-1650.7	1.8	-5677.5	
uncorr-s-w_01	375	14.9	c	1.00000	1.0	89.7	26.4		8.6	-205.8	15.3	-72.7	347.2	1213.8	24.3	-8.8	
uncorr-s-w_06	375	12.3	c	1.00000	0.3	46.8	165.9		3.5	-4625.3	5.4	-2999.6	15.7	-953.9	7.4	-2142.9	
uncorr-s-w_10	375	14.9	u	1.00000	0.2	17.0	230.9		2.1	-10782.5	2.4	-9387.6	4.1	-5482.5	2.9	-7988.5	
b-s-corr_01	375	0.0	c	1.00000	0.3	94.1	24.4	51.2	110.4	100.2	311.6	116.8	379.6	154.4	533.9		
b-s-corr_06	375	0.0	c	1.00000	0.3	56.6	83.5	149.0	78.3	56.8	-47.0	59.4	-40.6	98.2	17.6		
b-s-corr_10	375	0.0	c	0.99998	0.3	27.5	181.4	92.7	-95.7	26.7	-579.8	27.6	-556.8	37.8	-379.4		
uncorr_01	750	32.5	c	1.00000	0.4	71.4	92.1		21.3	-332.9	33.4	-175.7	131.9	43.3	52.8	-74.3	
uncorr_06	750	14.8	c	1.00000	0.2	39.0	429.8	19.6	-2089.8	16.1	-2564.3	40.0	-974.0	22.8	-1788.1		
uncorr_10	750	43.1	u	1.00000	0.2	13.0	306.7	3.8	-7956.5	3.4	-8792.8	10.4	-2848.0	3.6	-8485.1		
uncorr-s-w_01	750	16.7	c	1.00000	0.5	88.7	117.0		11.1	-950.9	19.2	-511.0	255.2	118.0	28.9	-304.5	
uncorr-s-w_06	750	13.5	c	1.00000	0.2	45.7	472.0		6.5	-7119.9	7.8	-5937.7	23.5	-1911.5	10.2	-4518.9	
uncorr-s-w_10	750	14.4	u	1.00000	0.3	17.0	823.7		4.8	-17214.5	5.1	-16043.7	10.8	-7520.6	5.6	-14608.0	
b-s-corr_01	750	0.0	c	0.99999	0.3	93.7	161.4	183.8	13.8	287.4	78.0	388.3	140.5	442.4	174.0		
b-s-corr_06	750	0.0	c	1.00000	0.5	55.4	259.6	975.0	275.6	175.8	-47.7	203.1	-27.8	155.2	-67.3		
b-s-corr_10	750	0.0	c	0.99999	0.2	25.9	281.7	20261.7	7091.7	176.7	-59.4	185.2	-52.1	136.3	-106.7		
instance family e1101																	
uncorr_01	100	49.0	c	1.00000	0.4	60.7	7.4		3.0	-144.7	6.0	-23.3	12.9	73.8	10.3	37.9	
uncorr_06	100	16.0	c	0.99993	0.3	39.7	20.5		2.4	-746.2	4.0	-410.0	8.3	-147.8	6.0	-242.2	
uncorr_10	100	57.0	u	1.00000	0.2	10.0	4.8		0.8	-524.7	1.0	-386.0	1.8	-164.5	1.2	-282.5	
uncorr-s-w_01	100	25.0	c	1.00000	0.3	90.3	6.8		4.0	-70.2	7.9	14.8	25.7	274.6	12.4	80.4	
uncorr-s-w_06	100	17.0	c	1.00000	0.4	41.9	17.2		1.6	-1001.7	2.6	-567.5	6.0	-186.2	3.8	-351.2	
uncorr-s-w_10	100	15.0	u	1.00000	0.2	17.2	39.6		1.0	-3748.5	1.3	-2963.4	2.1	-1761.6	1.6	-2318.5	
b-s-corr_01	100	0.0	c	1.00000	0.5	94.4	12.7	60.2	373.0	126.5	893.6	135.5	964.3	209.2	1543.2		
b-s-corr_06	100	0.0	c	1.00000	0.4	56.2	11.3		10.4	-8.5	22.9	102.3	22.9	101.9	40.1	254.2	
b-s-corr_10	100	0.0	c	0.99990	0.2	28.2	16.6		15.9	-4.5	27.3	64.3	27.0	62.0	41.6	150.1	
uncorr_01	500	38.8	c	1.00000	0.4	65.9	31.6		14.4	-120.2	28.0	-13.0	76.3	141.0	43.8	38.4	
uncorr_06	500	14.4	c	1.00000	0.3	39.2	397.3		8.3	-4678.5	13.7	-2810.3	26.3	-1408.1	19.6	-1927.6	
uncorr_10	500	51.4	u	1.00000	0.2	11.4	212.8		2.2	-9707.2	2.5	-8318.4	5.6	-3715.1	3.0	-6968.0	
uncorr-s-w_01	500	20.4	c	1.00000	4.5	88.4	88.7		22.5	-293.4	39.4	-125.1	1924.5	2070.4	60.2	-47.2	
uncorr-s-w_06	500	14.2	c	1.00000	0.4	44.8	365.2		5.3	-6825.2	7.7	-4638.1	20.7	-1661.9	10.7	-3317.2	
uncorr-s-w_10	500	16.4	u	1.00000	0.2	16.3	525.4		3.2	-16328.3	3.5	-14741.2	7.3	-7094.9	4.0	-13005.0	
b-s-corr_01	500	0.0	c	1.00000	0.4	93.5	64.6	433.8	571.5	624.6	867.0	773.4	1097.2	991.3	1434.6		
b-s-corr_06	500	0.0	c	1.00000	0.5	54.8	248.7	458.6	84.4	97.8	-154.4	101.6	-144.8	166.2	-49.7		
b-s-corr_10	500	0.0	c	0.9													



Table 2: Details on the CP Search Performed by BIB<sup>λ</sup> on Small Size Instances

instance	m	ver	BIB <sup>500</sup>			BIB <sup>1000</sup>			BIB <sup>1000</sup> <i>no seq.</i>			BIB <sup>1500</sup>		
			b	f	$UB^\lambda$ runs	b	f	$UB^\lambda$ runs	b	f	$UB^\lambda$ runs	b	f	$UB^\lambda$ runs
instance family e151														
uncorr_01	50	c	42	20	16 64 (0.25)	42	20	16 64 (0.25)	91	44	26 119 (0.22)	42	20	16 64 (0.25)
uncorr_06	50	c	34	17	17 65 (0.26)	34	17	17 65 (0.26)	72	35	31 101 (0.31)	34	17	17 65 (0.26)
uncorr_10	50	u	22	11	11 36 (0.31)	22	11	11 36 (0.31)	50	22	20 58 (0.35)	22	11	11 36 (0.31)
uncorr-s-w_01	50	c	46	21	14 84 (0.17)	46	21	14 84 (0.17)	163	79	41 199 (0.2)	46	21	14 84 (0.17)
uncorr-s-w_06	50	c	36	16	16 64 (0.25)	36	16	16 64 (0.25)	62	30	30 94 (0.32)	36	16	16 64 (0.25)
uncorr-s-w_10	50	u	8	4	4 33 (0.12)	8	4	4 33 (0.12)	26	12	12 54 (0.22)	8	4	4 33 (0.12)
b-s-corr_01	50	c	676	334	220 609 (0.36)	676	334	220 609 (0.36)	722	355	232 628 (0.37)	676	334	220 609 (0.36)
b-s-corr_06	50	c	100	50	50 139 (0.36)	100	50	50 139 (0.36)	104	51	51 141 (0.36)	100	50	50 139 (0.36)
b-s-corr_10	50	c	90	45	45 129 (0.35)	90	45	45 129 (0.35)	92	45	45 129 (0.35)	90	45	45 129 (0.35)
uncorr_01	250	c	110	52	50 254 (0.2)	110	52	50 254 (0.2)	293	138	97 472 (0.21)	110	52	50 254 (0.2)
uncorr_06	250	c	152	75	75 346 (0.22)	130	64	64 324 (0.2)	228	112	112 456 (0.24)	130	64	64 324 (0.2)
uncorr_10	250	u	36	18	18 145 (0.12)	36	18	18 145 (0.12)	151	74	73 264 (0.28)	36	18	18 145 (0.12)
uncorr-s-w_01	250	c	58	29	24 268 (0.09)	58	29	24 268 (0.09)	415	203	154 749 (0.21)	58	29	24 268 (0.09)
uncorr-s-w_06	250	c	48	23	23 255 (0.09)	48	23	23 255 (0.09)	175	86	85 415 (0.21)	48	23	23 255 (0.09)
uncorr-s-w_10	250	u	30	14	14 225 (0.06)	30	14	14 225 (0.06)	187	90	86 404 (0.21)	30	14	14 225 (0.06)
b-s-corr_01	250	c	600	296	265 1188 (0.22)	600	296	265 1188 (0.22)	684	338	305 927 (0.33)	600	296	265 1188 (0.22)
b-s-corr_06	250	c	1148	572	567 1388 (0.41)	662	331	331 885 (0.37)	700	350	350 920 (0.38)	444	222	222 670 (0.33)
b-s-corr_10	250	c	412	206	205 628 (0.33)	232	116	116 452 (0.26)	238	119	119 456 (0.26)	232	116	116 452 (0.26)
uncorr_01	500	c	1032	508	376 1080 (0.35)	964	470	354 1034 (0.34)	2292	1132	809 2482 (0.33)	932	455	344 1014 (0.34)
uncorr_06	500	c	332	164	163 755 (0.22)	266	131	131 683 (0.19)	610	301	300 1104 (0.27)	266	131	131 683 (0.19)
uncorr_10	500	u	36	18	18 285 (0.06)	36	18	18 285 (0.06)	269	131	128 548 (0.23)	36	18	18 285 (0.06)
uncorr-s-w_01	500	c	66	33	28 474 (0.06)	66	33	28 474 (0.06)	466	224	167 1160 (0.14)	66	33	28 474 (0.06)
uncorr-s-w_06	500	c	96	46	46 511 (0.09)	94	45	45 509 (0.09)	386	191	191 911 (0.21)	94	45	45 509 (0.09)
uncorr-s-w_10	500	u	38	17	16 442 (0.04)	36	16	16 440 (0.04)	382	187	187 872 (0.21)	36	18	18 440 (0.04)
b-s-corr_01	500	c	8318	4154	4061 9992 (0.41)	4486	2238	2148 6128 (0.35)	5310	2639	2528 5870 (0.43)	4486	2238	2148 6128 (0.35)
b-s-corr_06	500	c	21858	10918	10771 22543 (0.48)	2804	1396	1391 3305 (0.42)	3212	1601	1596 3694 (0.43)	1960	976	976 2453 (0.4)
b-s-corr_10	500	c	6402	3199	3167 6844 (0.46)	628	314	313 1102 (0.28)	650	324	323 1122 (0.29)	354	177	177 830 (0.21)
instance family e176														
uncorr_01	75	c	96	46	44 138 (0.32)	96	46	44 138 (0.32)	199	97	65 254 (0.25)	96	46	44 138 (0.32)
uncorr_06	75	c	56	26	26 107 (0.24)	56	26	26 107 (0.24)	91	44	44 149 (0.3)	56	26	26 107 (0.24)
uncorr_10	75	u	10	5	5 45 (0.11)	10	5	5 45 (0.11)	58	28	28 91 (0.3)	10	5	5 45 (0.11)
uncorr-s-w_01	75	c	110	54	33 170 (0.19)	110	54	33 170 (0.19)	338	167	86 433 (0.2)	110	54	33 170 (0.19)
uncorr-s-w_06	75	c	38	19	17 96 (0.18)	38	19	17 96 (0.18)	94	46	42 151 (0.28)	38	19	17 96 (0.18)
uncorr-s-w_10	75	u	20	10	10 74 (0.14)	20	10	10 74 (0.14)	65	29	28 121 (0.23)	20	10	10 74 (0.14)
b-s-corr_01	75	c	6218	3100	2684 6446 (0.42)	6218	3100	2684 6446 (0.42)	6856	3419	2993 6874 (0.44)	6210	3096	2680 6437 (0.42)
b-s-corr_06	75	c	110	55	55 178 (0.31)	110	55	55 178 (0.31)	128	63	63 194 (0.32)	110	55	55 178 (0.31)
b-s-corr_10	75	c	318	159	157 358 (0.44)	224	112	112 272 (0.41)	246	123	123 294 (0.42)	174	87	87 226 (0.38)
uncorr_01	375	c	212	106	102 438 (0.23)	212	106	102 438 (0.23)	473	236	178 785 (0.23)	212	106	102 438 (0.23)
uncorr_06	375	c	160	79	79 461 (0.17)	160	79	79 461 (0.17)	302	148	146 653 (0.22)	160	79	79 461 (0.17)
uncorr_10	375	u	46	22	21 230 (0.09)	46	22	21 230 (0.09)	545	268	260 718 (0.36)	46	23	23 230 (0.1)
uncorr-s-w_01	375	c	200	99	96 560 (0.17)	186	92	89 545 (0.16)	3624	1806	1386 4938 (0.28)	186	92	89 545 (0.16)
uncorr-s-w_06	375	c	78	38	37 437 (0.08)	78	38	37 437 (0.08)	266	127	126 661 (0.19)	78	38	37 437 (0.08)
uncorr-s-w_10	375	u	32	14	14 347 (0.04)	30	15	15 345 (0.04)	156	77	76 528 (0.14)	30	15	15 345 (0.04)
b-s-corr_01	375	c	568	278	227 1901 (0.12)	568	278	227 1901 (0.12)	662	325	265 1043 (0.25)	568	278	227 1901 (0.12)
b-s-corr_06	375	c	3360	1679	1667 3708 (0.45)	414	207	207 771 (0.27)	454	227	227 811 (0.28)	414	207	207 771 (0.27)
b-s-corr_10	375	c	5232	2610	2572 5543 (0.46)	586	290	289 935 (0.31)	668	331	330 1014 (0.33)	472	233	233 822 (0.28)
uncorr_01	750	c	488	243	238 991 (0.24)	352	175	170 851 (0.2)	1094	545	419 1924 (0.22)	352	175	170 851 (0.2)
uncorr_06	750	c	1254	624	610 1891 (0.32)	348	172	172 970 (0.18)	869	430	427 1610 (0.27)	348	172	172 970 (0.18)
uncorr_10	750	u	174	84	71 604 (0.12)	92	46	46 510 (0.09)	626	307	301 1091 (0.28)	58	29	29 475 (0.06)
uncorr-s-w_01	750	c	152	75	71 838 (0.08)	150	74	70 836 (0.08)	1668	824	630 3122 (0.2)	150	74	70 836 (0.08)
uncorr-s-w_06	750	c	136	65	62 803 (0.08)	74	36	35 736 (0.05)	384	191	187 1200 (0.16)	74	36	35 736 (0.05)
uncorr-s-w_10	750	u	28	13	13 677 (0.02)	22	11	11 667 (0.02)	550	274	271 1342 (0.2)	22	11	11 667 (0.02)
b-s-corr_01	750	c	2576	1282	1152 6124 (0.19)	1754	871	744 5281 (0.14)	2324	1150	975 3104 (0.31)	1754	871	744 5281 (0.14)
b-s-corr_06	750	c	39336	19666	19490 39884 (0.49)	1866	932	930 2597 (0.36)	2158	1077	1075 2892 (0.37)	686	343	343 1416 (0.24)
b-s-corr_10	750	c	1664464	832213	819235 1652903 (0.5)	4906	2450	2435 5635 (0.43)	5288	2640	2625 6034 (0.44)	1890	943	940 2615 (0.36)
instance family e1101														
uncorr_01	100	c	48	24	24 97 (0.25)	48	24	24 97 (0.25)	91	46	36 151 (0.24)	48	24	24 97 (0.25)
uncorr_06	100	c	90	42	42 161 (0.26)	90	42	42 161 (0.26)	151	72	72 228 (0.32)	90	42	42 161 (0.26)
uncorr_10	100	u	26	13	13 62 (0.21)	26	13	13 62 (0.21)	96	47	46 127 (0.36)	24	12	12 60 (0.2)
uncorr-s-w_01	100	c	44	21	12 121 (0.1)	44	21	12 121 (0.1)	149	73	34 269 (0.13)	44	21	12 121 (0.1)
uncorr-s-w_06	100	c	20	9	9 97 (0.09)	20	9	9 97 (0.09)	74	35	32 161 (0.2)	20	9	9 97 (0.09)
uncorr-s-w_10	100	u	22	11	11 105 (0.1)	22	11	11 105 (0.1)	67	31	30 158 (0.19)	22	11	11 105 (0.1)
b-s-corr_01	100	c	720	360	296 870 (0.34)	720	360	296 870 (0.34)	756	377	313 846 (0.37)	720	360	296 870 (0.34)
b-s-corr_06	100	c	144	72	72 237 (0.3)	144	72	72 237 (0.3)	152	76	76 245 (0.31)	144	72	72 237 (0.3)
b-s-corr_10	100	c	582	291	289 658 (0.44)	460	230	230 544 (0.42)	466	233	233 550 (0.42)	426	213	213 510 (0.42)
uncorr_01	500	c	200	100	98 494 (0.2)	200	100	98 494 (0.2)	485	242	199 872 (0.23)	200	100	98 494 (0.2)
uncorr_06	500	c	196	98	98 606 (0.16)	196	98	98 606 (0.16)	360	176	176 840 (0.21)	196	98	98 606 (0.16)
uncorr_10	500	u	48	22	20 282 (0.07)	44	22	21 278 (0.08)	242	119	107 486 (0.22)	44	22	22 278 (0.08)
uncorr-s-w_01	500	c	626	308	253 1098 (0.23)	612	301	247 1084 (0.23)	27178	13577	10687 35473 (0.3)	612	301	247 1084 (0.23)
uncorr-s-w_06	500	c	68	33	33 488 (0.07)	68	33	33 488 (0.07)	300	148	144 815 (0.18)	68	33	33 488 (0.07)
uncorr-s-w_10	500	u	28											

solution strategy might be to omit the pre-processing stage dealing with these instances in the case of  $LB_{red. pre-pr.}^{100}$  as its running time dominates that one of  $LB_{no pre-pr.}^{100}$ . However, the reasoning based on the sequencing constraints significantly improves the situation. It is able to reduce the time taken by this stage up to  $\sim 400$  times.

Table 3: Results of Computational Experiments on Large Size Instances

instance	m	$\alpha, \%$	ver	LB <sup>100</sup>		LB <sup>100</sup> <sub>no pre-pr.</sub>		LB <sup>100</sup> <sub>red. pre-pr.</sub>	
				t, sec	$\beta, \%$	$t_p$	$\omega, \%$	$\gamma$	$\eta$
instance family pla33810									
uncorr_01	33809	29.0	c	515	78.7	0	551	1.19	1
uncorr_06	33809	12.8	c	342	42.8	0	509	1.32	1
uncorr_10	33809	35.9	u	52	15.0	13	543	1.76	10
uncorr-s-w_01	33809	19.3	c	435	89.5	0	225	1.04	1
uncorr-s-w_06	33809	11.2	c	607	47.7	0	361	1.18	1
uncorr-s-w_10	33809	8.7	c	25	18.2	0	902	1.44	1
b-s-corr_01	33809	0.0	c	447	93.6	0	4	1.00	1
b-s-corr_06	33809	0.0	c	566	56.3	0	8	1.00	1
b-s-corr_10	33809	0.0	c	563	26.5	0	7	1.00	1
uncorr_01	169045	30.6	c	587	76.6	0	419	1.22	1
uncorr_06	169045	12.8	c	1204	42.7	0	91	1.32	1
uncorr_10	169045	35.8	u	157	14.7	94	-36	1.79	8
uncorr-s-w_01	169045	15.2	c	348	90.5	0	348	1.03	1
uncorr-s-w_06	169045	11.7	c	590	47.2	0	263	1.19	1
uncorr-s-w_10	169045	9.0	c	549	18.0	0	1037	1.46	1
b-s-corr_01	169045	0.0	c	1384	93.7	0	2	1.00	1
b-s-corr_06	169045	0.0	c	438	56.4	0	1	1.00	1
b-s-corr_10	169045	0.0	c	718	26.3	0	13	1.00	1
uncorr_01	338090	31.6	c	1589	75.4	0	486	1.24	1
uncorr_06	338090	12.8	c	1023	42.6	0	61	1.32	1
uncorr_10	338090	35.9	u	947	14.7	238	29	1.79	6
uncorr-s-w_01	338090	15.2	c	1209	90.5	0	324	1.03	1
uncorr-s-w_06	338090	11.9	c	966	47.1	0	111	1.19	1
uncorr-s-w_10	338090	9.0	c	1156	18.0	0	213	1.46	1
b-s-corr_01	338090	0.0	c	829	93.6	0	3	1.00	1
b-s-corr_06	338090	0.0	c	873	56.3	0	8	1.00	1
b-s-corr_10	338090	0.0	c	1095	26.3	0	7	1.00	1
instance family pla85900									
uncorr_01	85899	32.4	c	2514	73.8	0	322	1.27	1
uncorr_06	85899	13.5	c	3028	41.6	0	1230	1.35	1
uncorr_10	85899	40.8	u	213	13.7	59	52	1.92	28
uncorr-s-w_01	85899	16.4	c	1683	88.6	0	468	1.06	3
uncorr-s-w_06	85899	12.3	c	1985	46.6	0	575	1.20	1
uncorr-s-w_10	85899	13.6	u	158	17.2	3	637	1.53	393
b-s-corr_01	85899	0.0	c	3967	93.6	0	-2	1.00	1
b-s-corr_06	85899	0.0	c	1570	56.3	0	0	1.00	1
b-s-corr_10	85899	0.0	c	3491	26.4	0	4	1.00	1
uncorr_01	429495	32.5	c	3436	73.6	0	710	1.27	1
uncorr_06	429495	13.6	c	5495	41.4	0	757	1.36	1
uncorr_10	429495	40.4	u	1471	13.8	936	-58	1.90	18
uncorr-s-w_01	429495	16.3	c	2380	90.2	0	773	1.04	1
uncorr-s-w_06	429495	12.8	c	4508	46.3	0	140	1.21	1
uncorr-s-w_10	429495	13.2	u	545	17.4	23	141	1.51	256
b-s-corr_01	429495	0.0	c	3105	93.6	0	1	1.00	1
b-s-corr_06	429495	0.0	c	5483	56.2	0	0	1.00	1
b-s-corr_10	429495	0.0	c	4751	26.3	0	3	1.00	1
uncorr_01	858990	33.2	c	6904	72.6	0	74	1.29	1
uncorr_06	858990	13.6	c	5510	41.4	0	43	1.36	1
uncorr_10	858990	40.6	u	3650	13.9	1544	27	1.90	16
uncorr-s-w_01	858990	16.4	c	4859	90.2	0	786	1.03	1
uncorr-s-w_06	858990	12.7	c	7095	46.3	0	124	1.21	1
uncorr-s-w_10	858990	13.2	u	5058	17.4	68	154	1.51	183
b-s-corr_01	858990	0.0	c	5474	93.5	0	0	1.00	1
b-s-corr_06	858990	0.0	c	12566	56.2	0	1	1.00	1
b-s-corr_07	858990	0.0	c	13806	26.4	0	1	1.00	1

## 10. Conclusion

We have introduced a new non-linear knapsack problem where items to be selected are subject to the total reward that a vehicle obtains by summing up profits of chosen items and subtracting costs resulted from their transportation

along a fixed route. We have shown that both the constrained and unconstrained versions of the problem are  $\mathcal{NP}$ -hard. Our proposed pre-processing scheme can significantly decrease the size of instances making them easier for computation. The experimental results show that small sized instances can be solved to optimality in a reasonable time by any of the two proposed exact approaches. Larger instances can be efficiently handled by our approximate approach producing near-optimal solutions.

As a future work, this problem has several natural generalizations. The first evident generalization is for sure the traveling thief problem where the sequence of cities may be changed. This variant asks for the mutual solution of the traveling salesman and knapsack problems. Another interesting situation takes place when cities may be skipped because are of no worth, for example any item stored there has in fact low or negative contribution to the total reward. Finally, the possibility to pickup and deliver the items is for certain one another challenging problem. The outcomes of our research can further be adopted to solve routing problems with nonlinear cost functions, for example those when such a measure as gallon per vehicle mile versus load is used.

## 11. Acknowledgements

This research has been supported through ARC Discovery Project DP130104395.

## 12. References

### References

- Applegate, D., Cook, W. J., & Rohe, A. (2003). Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15, 82–92.
- Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, 19, 621–636.
- Bockmayr, A., & Hooker, J. N. (2005). Constraint programming. In G. N. K. Aardal, & R. Weismantel (Eds.), *Discrete Optimization* (pp. 559 – 600). Elsevier volume 12 of *Handbooks in Operations Research and Management Science*. doi:10.1016/S0927-0507(05)12010-6.
- Bonyadi, M. R., Michalewicz, Z., & Barone, L. (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013* (pp. 1037–1044). IEEE. doi:10.1109/CEC.2013.6557681.
- Bretthauer, K. M., & Shetty, B. (2002). The nonlinear knapsack problem - algorithms and applications. *European Journal of Operational Research*, 138, 459–472.

- Chekuri, C., & Khanna, S. (2005). A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, *35*, 713–728.
- Elhedhli, S. (2005). Exact solution of a class of nonlinear knapsack problems. *Oper. Res. Lett.*, *33*, 615–624.
- Erlebach, T., Kellerer, H., & Pferschy, U. (2001). Approximating multi-objective knapsack problems. In F. K. H. A. Dehne, J.-R. Sack, & R. Tamassia (Eds.), *WADS* (pp. 210–221). Springer volume 2125 of *Lecture Notes in Computer Science*.
- Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, *39*, 188–205. doi:10.1287/trsc.1030.0079.
- Garey, M., & Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
- GOODYEAR (2008). Factors Affecting Truck Fuel Economy. <http://www.goodyeartrucktires.com/pdf/resources/publications/FactorsAffectingTruckFuelEconomy.pdf>.
- Hochbaum, D. S. (1995). A nonlinear knapsack problem. *Oper. Res. Lett.*, *17*, 103–110.
- Li, H.-L. (1994). A global approach for general 0-1 fractional programming. *European Journal of Operational Research*, *73*, 590 – 596. doi:10.1016/0377-2217(94)90257-7.
- Lin, C., Choy, K., Ho, G., Chung, S., & Lam, H. (2014). Survey of green vehicle routing problem: Past and future trends. *Expert Systems with Applications*, *41*, 1118 – 1138. doi:10.1016/j.eswa.2013.07.107.
- Martello, S., Pisinger, D., & Toth, P. (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. *Manage. Sci.*, *45*, 414–424. doi:10.1287/mnsc.45.3.414.
- Martello, S., & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons.
- Polyakovskiy, S., Bonyadi, M. R., Wagner, M., Michalewicz, Z., & Neumann, F. (2014). A comprehensive benchmark set and heuristics for the traveling thief problem. In D. V. Arnold (Ed.), *GECCO* (pp. 477–484). ACM.
- Polyakovskiy, S., & Neumann, F. (2015). Packing while traveling: Mixed integer programming for a class of nonlinear knapsack problems. In L. Michel (Ed.), *Integration of AI and OR Techniques in Constraint Programming* (pp. 332–346). Springer International Publishing volume 9075 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-319-18008-3\_23.
- Reinelt, G. (1991). TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, *3*, 376–384. doi:10.1287/ijoc.3.4.376.

- Rossi, F., van Beek, P., & Walsh, T. (2008). Chapter 4 constraint programming. In V. L. Frank van Harmelen, & B. Porter (Eds.), *Handbook of Knowledge Representation* (pp. 181 – 211). Elsevier volume 3 of *Foundations of Artificial Intelligence*. doi:10.1016/S1574-6526(07)03004-0.
- Rossi, F., Beek, P. v., & Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc.
- Sherali, H., & Adams, W. (1999). *A Reformulation Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. J Kluwer Academic Publishing, Boston, MA.
- Tawarmalani, M., Ahmed, S., & Sahinidis, N. (2002). Global optimization of 0-1 hyperbolic programs. *Journal of Global Optimization*, 24, 385–416. doi:10.1023/A:1021279918708.
- Vansteenwegen, P., Souffriau, W., & Oudheusden, D. V. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, 209, 1 – 10. doi:http://10.1016/j.ejor.2010.03.045.
- Westerlund, A., Göthe-Lundgren, M., & Larsson, T. (2006). A stabilized column generation scheme for the traveling salesman subtour problem. *Discrete Applied Mathematics*, 154, 2212 – 2238. doi:10.1016/j.dam.2005.04.012. International Symposium on Combinatorial Optimization CO'02.